HUMS
Open Systems Specification

Goodrich Corporation

Simmonds Precision Products, Inc.

Fuel & Utility Systems

CAGE CODE 89305

	Prepared	Approved	Approved	Approved
By	W.Thomas	H.Clark	A. Duke	
Signed	W.Thomas	H.Clark	A. Duke	
Date	6/6/00	6/6/00	6/6/00	

REV	DATE	REV BY	PAGES AFFECTED	ECO NO.	REL NO.
INIT	5/26/00		Change document # from E-3424, include user guide and interface requirement spec information	E04486	00-1232E
A	5/22/02	WJT	All pages affected.	E08071	02-2161

Table of Contents

List of Figures

List of Tables

1 Scope

1.1 Identification

This document serves to detail the basic architecture of the Health Usage and Management System (HUMS) Integrated Mechanical Diagnostic Systems (IMDS) and its interfaces. The HUMS IMDS has been designed to be an open system, thereby allowing new health and diagnostic technologies to be integrated into the HUMS.

A HUMS technology provider may interface with this system by five distinct methods:

- Embedding software in the existing HUMS on-board system primary processing unit.
- Embedding software in the existing HUMS on-board system vibration processing unit.
- Use of a VME-based data acquisition and processing board within the HUMS on-board system.
- Interfacing a remote data acquisition and processing system with the HUMS on-board system.
- Providing software to run on the HUMS ground station (post-flight analysis).

1.2 HUMS System Overview

The HUMS is an integrated system designed to perform health and usage monitoring functions for fixed and rotary wing aircraft drive train, propulsion, and structural components. The complete HUMS system consists of the following elements:

- On-Board System (OBS)
- Ground Station (GS)

The OBS consists of the separate Line Replaceable Units (LRUs) as follows:

- Main Processor Unit (MPU)
- 0 or more optional Remote Data Concentrators (RDC)
- An optional display unit
- The Data Transfer Unit (DTU)

The OBS is responsible for collecting, processing, analyzing, and storing data obtained from sensors located throughout the aircraft. The MPU analyzes the input data for exceedances and events, calculates various flight regimes, performs various diagnostic algorithms, normalizes trend data, and stores the data to an onboard data cartridge. Two specific processors are utilized in the MPU. The primary processing unit (PPU) performs select data acquisition, processing, and communication with external interfaces. The PPU is supplemented with the vibration-processing unit (VPU), which performs high-speed data acquisition and processing of vibration (accelerometer) data. A user interface is provided via an on-board Control Display Unit (CDU) or other display devices connected through a data bus. This interface allows the operator to view aircraft operating data in real-time and provides password protected maintenance information. Exceedance alerts and aircraft status data to the aircrew is also provided. In addition, this interface also provides the aircrew with the appropriate prompts for sequencing through the diagnostic operations. The flight data is stored on a flash memory card.

The GS consists of 2 S/W configuration items that support the OBS:

- Ground Software (GS)
- MPU Software Loader/Verifier Utility

The GS is the primary user interface with the HUMS system. It is responsible for logging and maintaining all flight and maintenance data, generating aircraft maintenance-due lists based on flight data, performing aircraft configuration and parts tracking, generating engineering and management reports, and archiving data.

Specifically, the GS functions include:

- DTU Initialization and Download
- Parts and Maintenance Configuration Tracking
- Usage Calculations/Updates
- Condition Indicator Extraction
- Advanced Diagnostics
- Data Graphing, trending, and Reporting
- System/User Administration
- Interfacing to external Applications

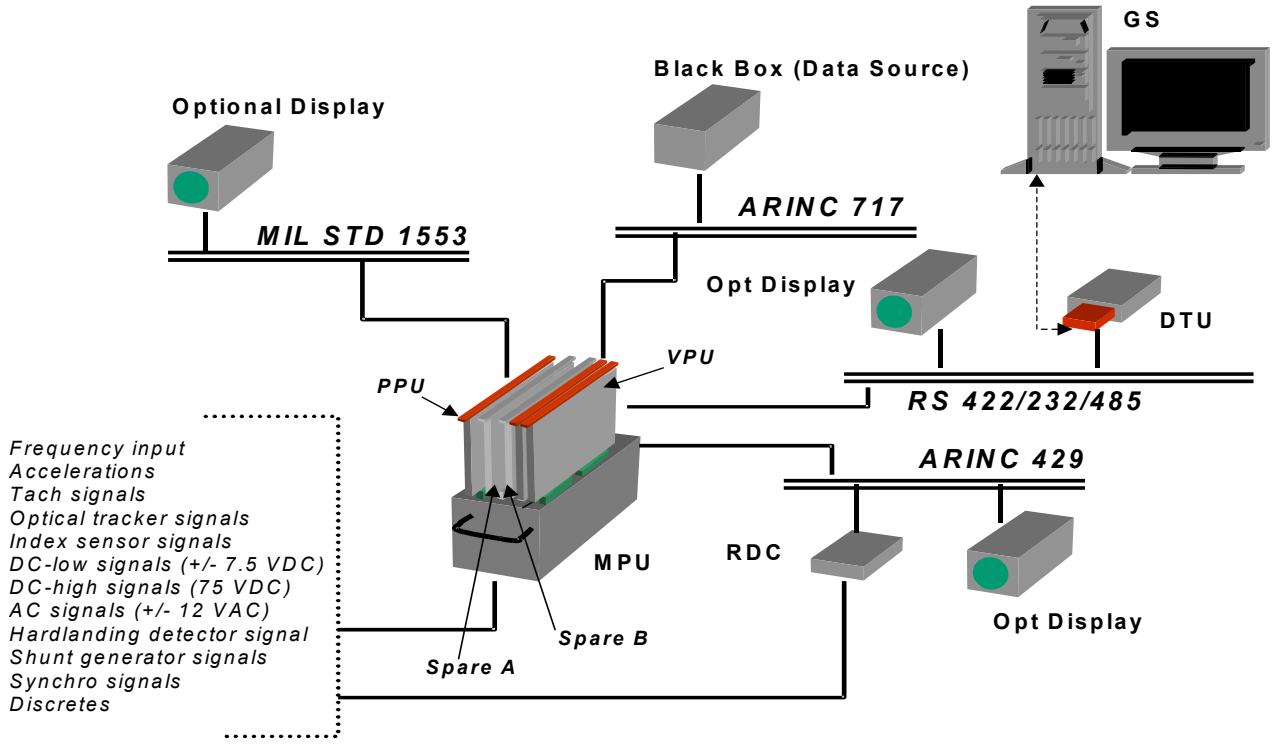


Figure 1-1 System Overview

1.3 Document Overview

This document specifies the Open System Interfaces of the HUMS IMDS system. This document consists of the following sections:

Chapter 1: Scope. Identifies the system and context of the system to which this document is written. Provides the context or system overview and the relationship between this document and others in the program.

Chapter 2: Applicable Documents. Provides a list of referenced documents.

Chapter 3: HUMS Open System Interface Summary. Provides a summary of all of the SW/HW interfaces that are provided to third party developers in the HUMS system.

Chapter 4: PPU Embedded Software. Provides a detailed description of the Ada95 interfaces available on the Onboard System

Chapter 5: VPU Embedded Software Interfaces. Provides a detailed description of the 'C' interfaces available on the VPU.

Chapter 6: VME Board Interface. Provides a description of the characteristics and parameters associated with the hardware interface of the VME board.

Chapter 7: MPU External Interfaces. Provides a description of the available I/O and the mechanical / electrical characteristics of the MPU's external interfaces.

Chapter 8: Ground Station Interfaces. Provides a reference to the proper GS document.

Chapter 9: HUMS System Integration Process Model. Defines responsibilities of third party developers and describes typical integration scenarios.

Chapter 10: Notes. Provides a list of abbreviations and acronyms.

Appendix A: MPU External Connector Signal Assignments. Provides all of the pin assignments of the external ARINC connectors of the MPU.

Appendix B: Spare Board Signal Assignment & Geometry. Provides all of the signals and pin assignments for the spare VME connector in the HUMS system.

Appendix C: Technology Integration Questionnaire. Provides a set of questions that third party developers answer and submit to Goodrich; Used to asses the viability of the technology being considered for integration.

1.4 Document Control

This document is provided by and maintained by Goodrich Corporation. The information in this document is subject to change without notice and should not be construed as a commitment by Goodrich Corporation. Goodrich assumes no responsibility for any errors that may appear in this document.

2 Applicable Documents

The following documents of the exact issue shown form a part of this document to the extent specified herein. In case of a conflict between the documents referenced herein and the contents of this document, the contents of this document will shall take precedence.

Note: In an attempt to make this document applicable to any aircraft on which a HUMS systems is installed Goodrich has factored out, as much as possible, all references to aircraft specific documentation. In the body of this document, references to aircraft specific documents will be made to titles where the actual aircraft identifiers are replaced by the phrase "*Aircraft Specific*" in the title. An example would be: *Aircraft Specific IMDS Core Parameters* document.

2.1 Customer Documents

NA

2.2 Goodrich Documents & Drawings

Document Number	Document Title
6000051-01-ICD-0101	Interface Control Document for the Health and Usage Management System Activity Data File Component
6000051-45-ICD-0101	Interface Control Document for the HUMS Task Controller Component
30190-0458-01	Circuit Card Assembly: Spare Board Layout Drawing

2.3 Other Documents

Document Number	Document Title
ARINC-429-14-93	Mark 33 Digital Information Transfer System (DITS) (includes supplements 1 through 14)
EIA RS-232-E-91	Interface between Data Terminal Equipment and Data Circuit Terminating Equipment Employing Serial Binary Data Interchange
EIA RS-422-B-94	Electrical Characteristics of Balanced Voltage Digital Interface Circuits
IEEE Standard 1014-1987	IEEE Standard for a Versatile Backplane Bus: VME bus (ANSI)
IEEE Standard 1101.2-1992	IEEE Standard for Mechanical Core Specifications for Conduction-Cooled Eurocards
MIL-STD-1553	Digital Time Division Command/Response Multiplexing Data Bus, Revisions A & B
MIL-STD-461D	Control Of Electromagnetic Interference Emissions and Susceptibility, Requirements For The Control Of
RTCA / DO-160C-1989	Environmental Conditions and Test Procedures for Airborne Equipment
ISO/IEC 8652:1995	Ada95 Reference Manual: Language and Standard Libraries
ISO 8601:2000	Data elements and interchange formats -- Information interchange -- Representation of dates and times.
UD/REF/A1910-057020/002	Aonix Language Reference Manual
RTCA/DO-178B	Software Considerations in Airborne Systems & Equipment Certification (1,12,1992)

3 HUMS Open System Interface Summary

This section provides a brief description of the various types of open system interfaces available in each of the major functional components of the HUMS system.

3.1 PPU Software Interface

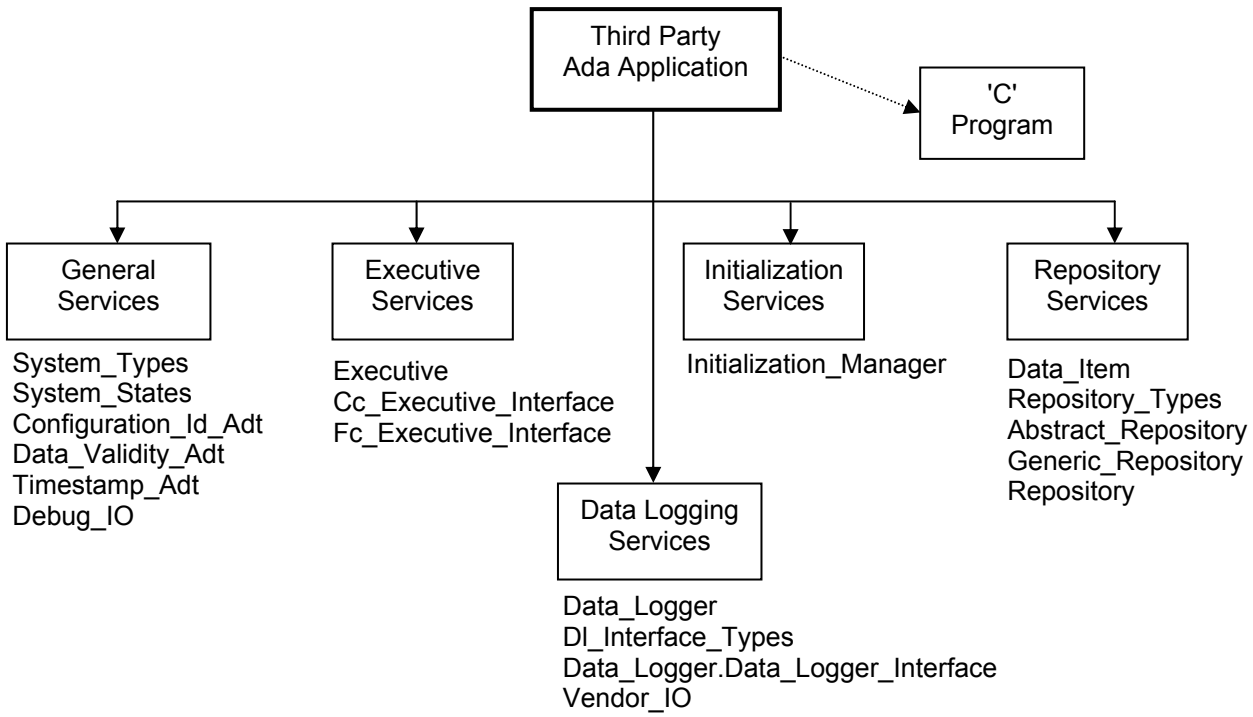


Figure 3-1 PPU Software Interface Overview Diagram

- Third Party Ada Application

Third party technology providers provide software applications written in the Ada95 language. The main interfaces (services) available to developers come in the form of Ada95 packages.

- Interfaces to Other Languages

Third party technology providers have the ability to incorporate software written in the 'C' language. The 'C' language modules do not have direct access to the services, instead an Ada wrapper supplies all data and I/O required by the module.

- General Services

The General Services include all of the general type packages; a package for working with data validity attributes, a package for working with data timestamps, and a package that provides general debug output for use during target integration.

- Executive Services

The Executive Services provide applications with the capability to register for and receive cyclic updates. It provides the main thread of control for the software applications.

- Initialization Services

The Initialization Services provide applications with the capability to read configuration data.

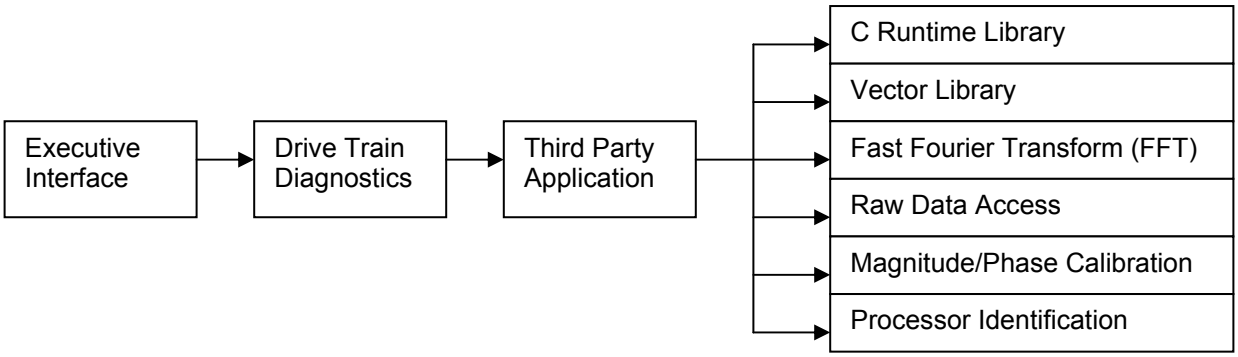
- Repository Services

The Repository Services provide applications with the capability to read and write values from the On-Board Systems internal data store.

- Data Logging Services

The Data Logging Services provide applications with the capability to log data to the On-Board Systems external data store, the Data Transfer Unit

3.2 VPU Software Interface



3.3 External MPU Hardware Interfaces

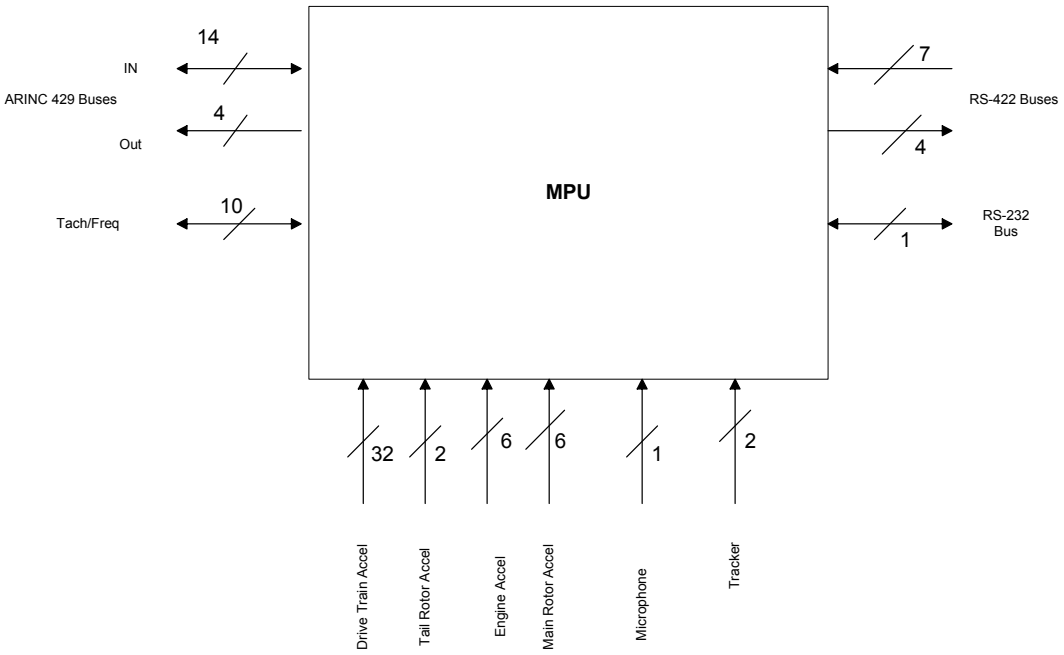


Figure 3-2 MPU Hardware Interfaces

3.4 Ground Station Software Interfaces

3.4.1 File / Data Interfaces

The Ground Station application stores flight data in two binary data files. The Raw Data File (RDF) is a copy of the flight data as it is stored by the On-Board System (OBS) on the flash memory card. The Activity Data File (ADF) is an index file into the flight data contained within the RDF. Since the format of these two files is under the control of Goodrich Corporation, the method by which a third-party software application can access raw flight data is through the Ground Station's Activity Data File Component. Using this component the format of the RDF and ADF is hidden from the software developer, and the data contained within the files is exposed as a hierarchy of Common Object Model (COM) objects that are intuitively organized to permit a "drill-down" approach to retrieving data. See the *Interface Control Document for the HUMS Activity Data File Component* for more information.

3.4.2 Application Architecture Interfaces

The Ground Station application architecture includes frameworks by which extension of the base functionality can be achieved. Goodrich Corporation utilizes these frameworks internally to increase the flexibility and configuration options of the Ground Station, which typically must be customized for each installation. A third party technology developer can utilize these interfaces to extend the Goodrich Ground Station to perform functionality adjunctive to HUMS. Specifically, the HUMS Task Controller Component of the Ground Station supports the IHUMSDownloadTask and IHUMSDownloadTask2 interfaces, which are the interfaces required to be supported by components that are run during an aircraft operation download. See the *Interface Control Document for the HUMS Task Controller Component* for specific details.

4 PPU Embedded Software

4.1 CSCI Description

The Flight Program CSCI is part of the Main Processor Unit. Figure 4-1 depicts the relationship between this CSCI and the other CSCIs that reside in the Main Processor Unit.

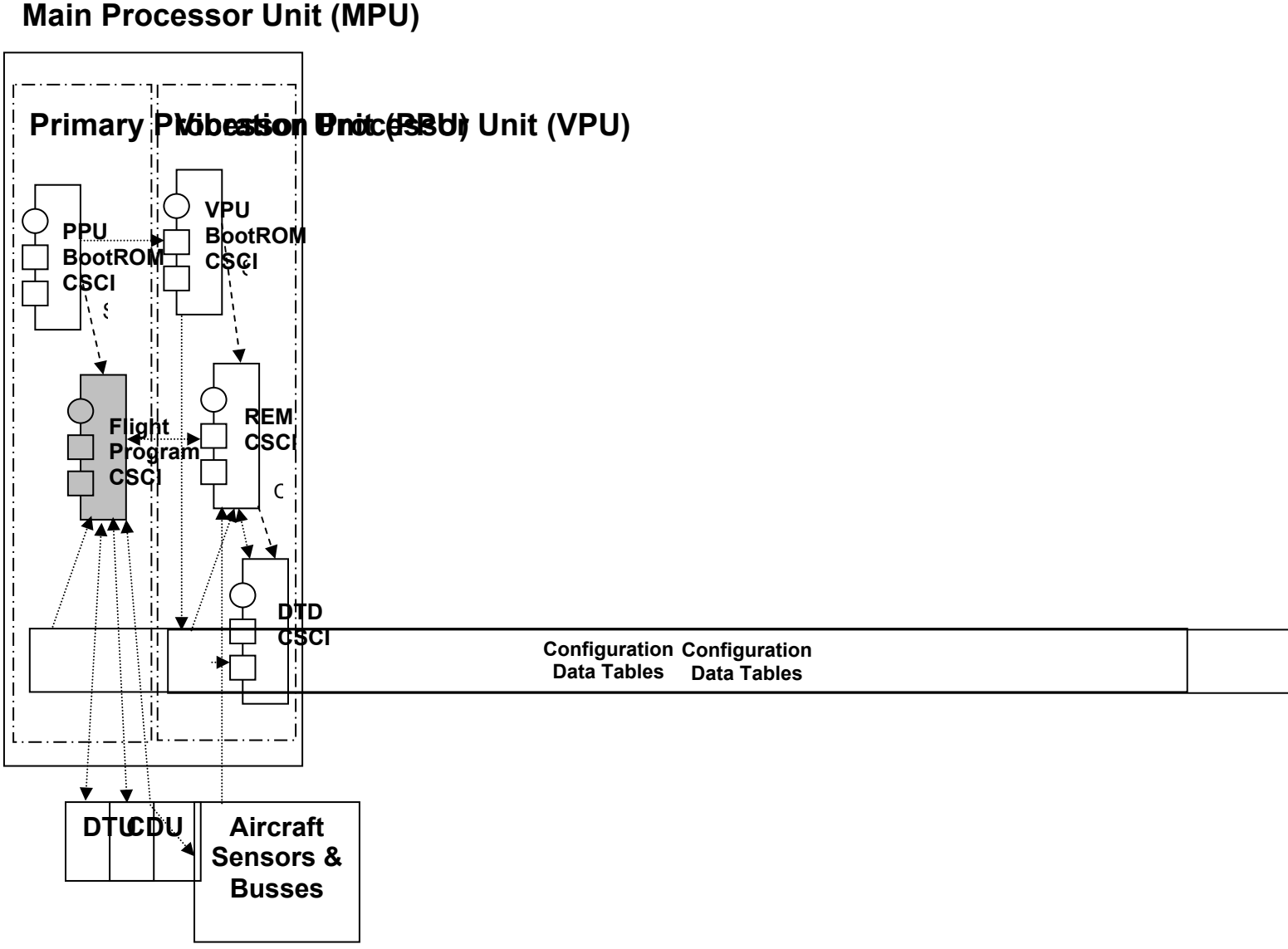


Figure 4-1 Main Processor Unit CSCIs

4.1.1 System States and Modes

State	Mode	Description
Initialization	Cold Start	The Cold_Start Mode (Sub-State) will perform the entire suite of power up operations including built-in test. This mode will also determine if a new configuration data load has been placed into Flash memory since the last cold start. If new data has been placed into Flash, processing will occur. Upon completion of the cold start operations, a transition to the 'Warm_Start' Mode will occur.
	Warm Start	The Warm_Start Mode (Sub-State) will perform the normal system and object initialization. This is the mode where the internal software objects will be created, initialized, and bound together to define the system. Upon completion of Warm_Start, the system will transition to the 'Normal_Processing' State.
Normal	--	The Normal Mode will allow data acquisitions to be performed and allow data logging to the DTU device.
Shutdown	--	The Shutdown Mode will perform the operation of placing all state data required to allow restart of the system to the current configuration into non-volatile RAM. When particular criteria has been met, the software will initiate a hardware shutdown sequence and then terminate.

Table 4-1 System States

4.1.2 Memory Resources

The PPU memory consists of 1Meg bytes of battery backed up RAM (BBRAM), 32M bytes of dynamic RAM (DRAM) and 8M bytes of Flash (ROM). The memory allocation for the Boot CSCI, Flight Program CSCI and Configuration Data will vary between aircraft models. Contact Goodrich for memory allocation profiles for a particular aircraft.

4.1.3 CPU Resources

The PPU Flight Program CSCI executes on a PowerPC 603e 32-bit microprocessor operating at 128 MHz.

4.1.4 System Timing

Typically, the PPU is supporting 0.2 to 20 Hz processes (0.05 to 5 sec). There is a 50% spare provision within the PPU. This implies that the PPU must perform all operations within 2.5 seconds of the 5-second major frame. For a simple (best-case) HUMS application, a third-party technology provider could have up to 25% of the available processing time (0.625 sec of the 2.5 sec major frame).

4.1.5 Data Flow

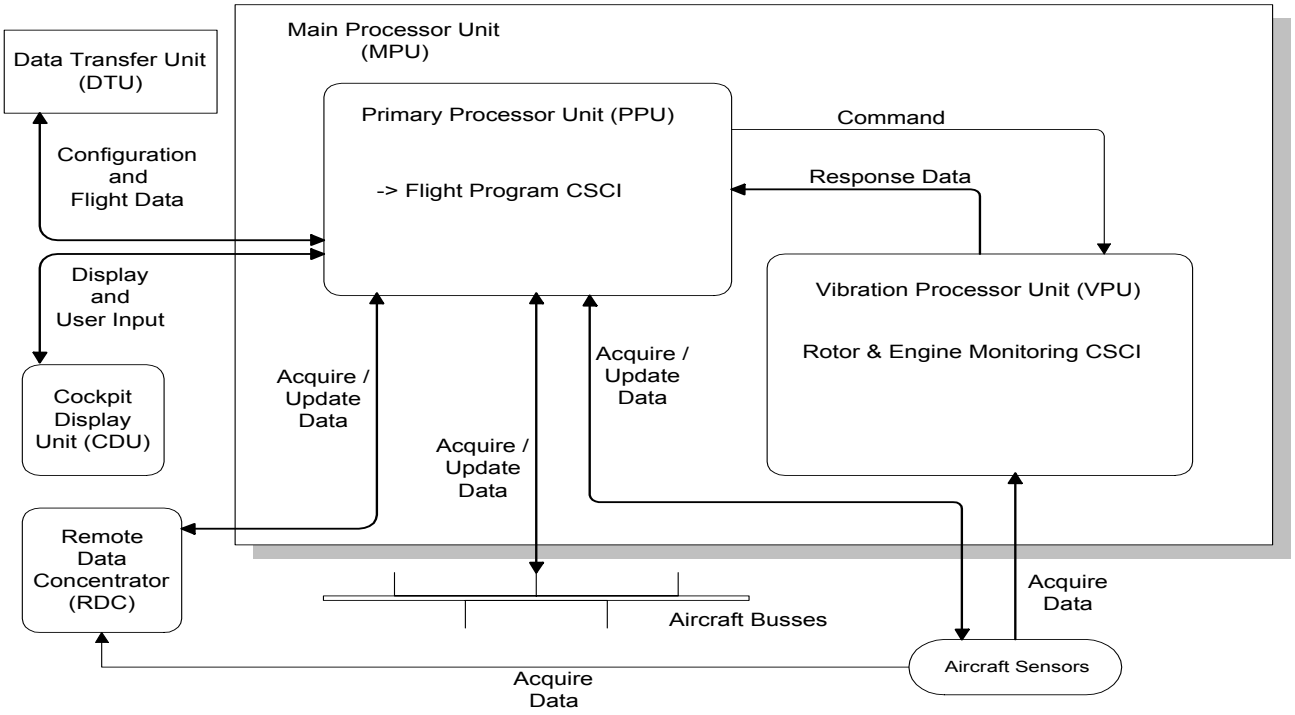


Figure 4-2 PPU Data Flow Diagram

4.2 PPU Interfaces Overview

4.2.1 Development Platform

The Flight Program CSCI utilizes functions/classes defined within the ObjectAda Real-Time Windows NT x PowerPC/RAVEN™ subset of the *Ada 95 Language Reference Manual*. Third-party developers are advised to review Aonix document UD/REF/A1910-05720/002 for specific descriptions of supported functions/classes from the Ada 95 Language Reference Manual Annexes.

The Flight Program CSCI also has the ability to interface to object modules written in the 'C' language. The 'C' compiler used must be the one that is released with the ObjectAda Real-Time Windows NT x PowerPC/RAVEN™ toolset.

4.2.1.1 General Notes on Developing Interface Modules in Ada95

The Aonix Ada95 compiler (which conforms to the RAVENSCAR profile) does not provide a default heap management system for dynamic acquisition of memory via allocators. Therefore, all variables must be statically allocated. Keep in mind that this does not prohibit the use of 'general' access types that can provide access to aliased (statically allocated) objects.

4.2.1.2 General Notes on Developing Interface Modules in 'C'

C modules (developed with the above mentioned 'C' compiler) should always be compiled and delivered as a pure object module ready for linking. When archiving C code as a library, use the 'ar' command (archive command) which produces a pure object file that can be linked with the main application. Do not use the 'ld' command that produces an executable image.

Just as there is no dynamic memory allocation of data in Ada95 there is also none available in 'C'. All memory must be statically allocated. Note that the 'C' compiler does not initialize static memory areas to any specific value. It is the responsibility of the developer to initialize all statically allocated memory before use.

4.2.2 General Description of Interface Objects

The HUMS Flight Program CSCI is based on an object oriented design methodology. The design is realized in Ada95, which is an object-oriented language. In general, objects are instances of classes. Classes in a Ada95 are realized using the Package/Type mechanism. Packages are the modules in which tagged types and primitive subprograms are declared that define the functional behavior of objects of the tagged types. Objects are simply variables of the tagged types; the variables are passed as parameters to the primitive operations to achieve the desired behavior. New classes can be derived from existing classes and new members or components can be added to the objects, new primitive operations can be defined as well.

Third party vendors will use the existing classes that are defined in the HUMS Open Systems Interface to create objects of their own that execute in the HUMS system. Third part classes will derive new subclasses from the existing HUMS classes and they will develop associations to objects created from these classes. For instance a third party class will derive from existing executive classes in order to provide operations that require periodic scheduling; they will then create an aggregate of internal objects for access to Configuration Data, Repository Items and Data Logging Objects.

4.2.3 Description of Configuration Data

The HUMS system is built around the concept of a “Data Driven Architecture”. Most of the systems run time structure and behavior is controlled by data tables that are resident in memory and read at system initialization time. The main concept adhered to during system design was to factor out as many system parameters as possible and to make them “configurable parameters”. Configurable parameters drastically reduce the need for code modifications when adapting the HUMS system to a new aircraft. In most cases, the system can completely adapt to a new aircraft by simply changing the configuration data tables.

Users of the OS Interface may also take advantage of the concept of “configuration data/configuration parameters” when designing their software classes/objects. During the design phase the user should be looking for ‘structural/behavioral dimensions and attributes’ that have a high probability of changing, then factoring them out into configuration parameters. Keep in mind that configuration data files can be loaded onto the target hardware without having to reload the system software.

Goodrich has allocated several groups of Configuration Ids (Data Table Ids) for third party vendors. The Configuration Ids manifest themselves as 32 bit unsigned integer values. Each 32-bit double word value is comprised of two 16 bit words. The most significant word represents the Class Id value; the least significant word represents the Instance Id. Each vendor is assigned a maximum of 10 Class IDs that limits the vendor to ten main classes of objects within the system. Goodrich uses the convention that the first number in class 1 is the developers 'main' number that is used to schedule the developers object. Within each class, the vendor is limited to 512 specific instances of those classes. An example of a vendor class could be a Sensor, where the vendor may define 12 different sensors objects that would correspond to 12 specific instances of the vendors Sensor class. The vendor could then define configuration data for each sensor and access that data with a unique Configuration Id all having the same Class component but different Instance Ids.

4.2.3.1 Private Configuration Data verses Reference Lists

Third party vendors can define their own private configuration data that they have access to using the Vendor Id's described above. The vendor can also specify that Goodrich place a list of configuration id's, a reference list, to other system entities at one of the vendors Id's. This reference list is the mechanism that vendors use to gain access to repository items and other systems objects it may require. In order for Goodrich to supply this reference list of configuration data the vendor must supply Goodrich with a list of nomenclature corresponding to the data items needed.

4.2.3.2 Configuration Data Generic Data Packet

Users of the OS Interface must determine the format of the configuration data and submit the table data values in the form of a binary data file to Goodrich prior to system integration. The binary data file must be formatted according to the generic data packet definition below (see Table 4-1), keep in mind that the data file requires a big endian byte gender (bit 0 is the MSB). Goodrich will then include these user defined data tables into its main configuration data file to be available to the user at system initialization time. The vendor can freely utilize the Cyclic Redundancy field; Goodrich will not validate the field.

Offset (Bytes)	Length (Bytes)	Name	Type	Description
0	4	Class Identifier	UDWORD	Unique value supplied by Goodrich. Contains only the Class Id, the Instance is zero.
4	4	Length	UDWORD	The number of bytes in length of the entire data packet, including the Identifier and CRC.
8	ND	Data Item		Vendor specific data.
8 + ND	NP	Pad	UBYTE(s)	Variable length field that forces the CRC data element to be on a 32 bit boundary. The value of all pad bits is zero.
8 + ND + NP	4	CRC	UDWORD	The Cyclic Redundancy

Table 4-2 Generic Data Packet Format

4.2.4 Periodic Scheduling

The HUMS system schedules objects for execution by using the `Executive` subsystem. The `Executive` subsystem provides separate threads of control, each executing at a configurable rate (predetermined by Goodrich). The `Executive` subsystem provides the subprograms needed to register system objects to be periodically scheduled. The package `Cc_Executive_Interface` provides an abstract base class that defines all of the primitive operations needed by any user derived class that has scheduling needs.

The heart of the `Cc_Executive_Interface` functionality is an abstract procedure named `Update`. The procedure `Update`, redefined by the user, is the procedure that `Executive` dynamically dispatches to during run-time at some predetermined rate. The body of `Update` (written by the user) will perform calls to all of the application specific cyclic processing routines needed within the users subsystem. The user-defined class must register a reference to its object with the `Executive` at system initialization time.

A fixed number of scheduling rates are provided, with the rate specified in configuration data. Configuration data will also specify the rate at which each schedulable object in the system must called. For example the rates configured may be; 1Hz, 2Hz, 5Hz, 10Hz, 20Hz, and 30Hz. Contact Goodrich for a list of rates for a specific aircraft.

4.2.5 Data Repository

The Data repository provides the isolation of the core Flight Program software from the specifics of the aircraft configuration and the details of the hardware interfaces.

The Data Repository serves as a standard method of communicating data from one object to another. Instead of having objects communicate directly with one another to pass data, they can use the data repository to pass data. This eliminates unnecessary object-to-object dependencies.

The Data Repository has been designed to allow single producers and multiple consumers of specific data objects. Only one producer can be registered for any given value. The Data Repository is the internal data store of the HUMS system. It provides the common interface to all aircraft data being monitored or generated by the Flight Program software.

The Data Repository has no knowledge of producers, consumers, or any other client of the Repository. It maintains data for its clients based on pre-determined initialization parameters. After a client stores data or retrieves data, the client and the repository each have their own copy of the data. The Repository makes no assumptions about how clients will use data.

4.2.5.1 Events and Exceedances in the Repository

Events in the system are considered to be either Monitored or Instantaneous. Instantaneous events are generated by objects/processes under unique circumstances and they exist only for a particular instance of time. The only action the system performs on instantaneous events is to log them to the DTU card. They are never assessable to any other object or process within the system. Monitored events on the other hand last for some duration of time and as such, their state can be reflected in a repository value.

There are a number of predefined exceedances/events listed in the Goodrich Document E-#### *Aircraft Specific IMDS Core Parameters.* Third party developers can also specify their own monitored events (in the Technology Questioner Appendix C). Goodrich will then define a boolean (discrete) repository item that will reflect the state of the event/exceedance and make the configuration id associated with that discrete available to the developer.

4.2.6 Data Logging

Third party developers can log data to the DTU card in the form of private data packets. This can be accomplished using the package `Vendor_IO`. The private data packets can contain any type of user data; the data is logged as an array of bytes. The developer logs data by calling the routine `Log_Private_Data`. The routine accepts an array of bytes up to a max size of 4096 bytes, a Cage Code, and a Vendor ID. The packets can be retrieved on the ground station using the Cage Code as a key.

The packets themselves are not explicitly sequenced; it is the responsibility of the third party developer to devise a scheme for sequencing the packets. This will usually take the form of a sequence number embedded within the data itself. Absolute timestamps are another data item that the developer may embedded in the data to add context.

There is a throughput limit of 115K bits per second going to the DTU. This will factor into the amount of data that a developer write to the card on a cyclic basis.

Note: It is a good practice to develop applications that write their data periodically to the DTU card instead of waiting for a possible "once a flight event" such as engines off, or landing. There is no telling if a particular condition will ever be met in flight, so it is a good practice to accumulate data over time. This will ensure some data is recorded even if the "flight event" is not detected.

4.2.7 System Initialization

System initialization is a major phase of the systems operation. The system works on a convention that each class of active system objects (ie. sensors, displays…) will be associated with one factory (see Figure 4-4 Executive Factory Initialization Sequence) that performs the initialization sequence particulars for all objects within that class. System initialization occurs in three separate phases. They are the ***Creation Phase***, ***Reference Resolution Phase***, and the ***Hardware Initialization Phase***.

System initialization is accomplished through a specific class defined as the `Cc_Factory_Interface` class. This class defines three main procedures `Create`, `Resolve_References`, and `Initialize_Hw` that correspond to the three phases of initialization. Prior to system initialization, during 'package elaboration' the factory objects are 'registered' with the system `Executive` class. During initialization each of the three defined procedures of the factory class are called by the executive during their appropriate phase. The factory procedures then in turn perform initialization on each of the class objects individually.

4.2.7.1 Creation Phase

During the ***Creation Phase*** of system initialization, all of the objects (instances of classes) are created and are initialized with their configuration data. In the objects configuration data there is usually Configuration Ids that will be used to gain access to other objects that will be needed during normal operation. For example, all of the "Repository Entries" that an object will need are received first as Configuration Ids in the objects configuration data; during the Reference Resolution phase the retrieved configuration ids will then be used to obtain references or pointers to the actual repository entries.

4.2.7.2 Reference Resolution Phase

During the ***Reference Resolution Phase*** of system initialization, all the objects in a class use the Configuration Ids they received in their configuration data and call various routines in the system that take configuration ids as inputs and yield references to other system objects. A user will find that the class factory is always the best place to retrieve object references since a factory is responsible for the creation of all objects of a particular class. The required factory is then the logical candidate to store and dispense the references (pointers) to the objects it has created.

So for instance if the system contained a Sensor Manager object which had to be associated with a number of Physical Sensor Objects then during the Reference Resolution phase the Sensor Manager object would call the Physical Sensor factory to obtain a reference or pointer to all of the Physical Sensors objects that it created. The Physical Sensor factor would then require a routine in its interface that takes in a Configuration ID and then returns a pointer to a Physical Sensor object.

The repository itself creates all of the repository items. Therefore, if a particular object needs access to a repository item then they call one of the `Repository` procedures `Register_as_Consumer` or `Register_as_Producer`, input a Configuration ID to it and then obtain out a pointer to the repository item.

Keep in mind that object references are obtained only after the Creation phase takes place, this is in order to eliminate any race conditions by guaranteeing that all objects have been created and that all references will exist.

4.2.7.3 Hardware Initialization Phase

The Hardware Initialization phase is the last in the initialization sequence. This phase is entered after all system level objects are created and after all references are resolved. This is the time in which to accomplish any low level hardware initialization required by the objects.

4.3 PPU Interfaces Detailed Package Description

The software inventory associated with this Open System Software Interface consists of a set of Ada packages. The specific Ada packages are listed below along with a short description of the purpose and functionality. The only other software packages that the Software Interface depends upon are the predefined Ada packages that are defined as part of the Ada compilation system that is described in the section entitled *Software Environment*.

Package Name	Description
System_Types	Defines general-purpose types used throughout the system.
System_States	Defines a set of system states that are used in determining the status of the system.
Initialization_Manager	The functions used to retrieve configuration data tables from memory.
Data_Validity_Adt	The operations used to manipulate objects that reflect the validity of various data objects.
Configuration_Id_Adt	Defines a set of data types and operations that define the unique identifiers of predefined groups of configuration data.
Cc_Executive_Interface	The base class used for all objects that require to be scheduled periodically.
Fc_Executive_Interface	The base class for the definition of specific factories that initialize sets of objects at system initialization time.
Executive	The subsystem used for scheduling objects that have registered for periodic processing services.
Timestamp_Adt	The class used for objects that manipulate system absolute and relative timestamps.
Data_Item	The base type of all data items that will be stored in the data repository.
Repository_Types	Simple package defining a string type and blank string constant.
Abstract_Repository	The abstract base class to be used as the foundation for all specific repositories.
Generic_Repository	A Generic package that allows particular types of items to exist in the repository.
Repository	The Repository Object includes instances of the generic repositories and includes general repository operations.
Data_Logger	The parent package for the data logging routines.
Dl_Interface_Types	Defines the types used with the Data Logger Interface
Data_Logger.Data_Logger_Interface	The main interface into the data logging functions.
Vendor_IO	Package that provides an interface to log private data packets.

Table 4-3 PPU Interface Packages

4.3.1 Overview of Derivable Base Types

Classes in Ada take the form of type definitions declared in Ada package specifications. The following is a discussion of three main type definitions that are found in many of the Ada packages that define classes in the HUMS system. The type `Object` is an abstract tagged limited type that means that it is an extendable record type (you can add components through derivation) and its inner details are private. Since it is limited private, objects of this type cannot be copied via assignment, they can only be manipulated through the primitive operations that are defined in the specification of the package.

The type `Reference` is an access type (pointer type) that can be used to supply indirect access to objects of any type rooted at the type `Object`. Values of this type can point to objects of type Object and this value can be used to pass the object to subprograms that both read and write the internal values of the data type.

The `View` type on the other hand defines an access type that provides read only access to the object, so values of this type can only reference objects and pass them to procedures that read the internal components of the type.

```
type Object    is abstract tagged limited private;
type Reference is access all      Object'Class;
type View      is access constant Object'Class;
```

4.3.2 Package System_Types

The package `System_Types` defines a set of general-purpose types that are used throughout the Open System Interfaces. It is declarative in nature in that it only provides type definitions.

```
with Interfaces;
package System_Types is -- ================================================

    type Byte_Array           is array (Integer range <>) of Interfaces.Unsigned_8;

    type Byte_Array_Ref       is access all Byte_Array;

    type Ieee_Float_32_Buffer is array (Integer range <>) of Interfaces.Ieee_Float_32;

    type Ieee_Float_64_Buffer is array (Integer range <>) of Interfaces.Ieee_Float_64;

    type Integer_16_Buffer    is array (Integer range <>) of Interfaces.Integer_16;

    type Integer_32_Buffer    is array (Integer range <>) of Interfaces.Integer_32;

    type Return_Status is (Ok, Failed, Unknown);

end System_Types; -- ========================================================
```

4.3.2.1 Package System_Types Type Declarations

The type `Byte_Array` is an unconstrained array type definition of 8 bit unsigned components indexed by the predefined type Integer. The type `Byte_Array_Ref` is an access type (pointer type) that can be used to reference objects of type `Byte_Array`.

The types `IEEE_Float_32_Buffer` and `IEEE_Float_64_Buffer` are unconstrained array type definitions of 32-bit and 64-bit floating-point components respectively; the predefined type Integer indexes both array types.

The types `Integer_16_Buffer` and `Integer_32_Buffer` are unconstrained array type definitions of 16-bit and 32-bit unsigned integer components respectively; the predefined type Integer indexes both array types.

The type `Return_Status` is an enumeration type with the values `Ok`, `Failed`, and `Unknown`. It is used in the declaration of several status parameters in various subprograms in the package `Executive`.

4.3.3 Package System_States

The package `System_States` defines a set of types and constants that are used throughout the Open System Interfaces when specifying or inquiring about the state of various system operations. It is declarative in nature in that it only provides type definitions. A detailed explanation of the types is as follows.

```
with Interfaces;
package System_States is -- ================================================

    type Initialization_Mode is (Create_Objects,
                                 Resolve_References,
                                 Initialize_Hw);


    subtype System_State     is Interfaces.Unsigned_8;

                  -- ==================================--

    Initialize  : constant := 2#0000_0001#;
    Normal      : constant := 2#0000_0010#;
    Shutdown    : constant := 2#0010_0000#;

end System_States; -- ======================================================
```

4.3.3.1 Package System_States Type Declarations

The type `Initialization_Mode` is an enumeration type with the values `Create_Objects`, `Resolve_References`, and `Initialize_Hw`. This type defines the three different types of initialization that exist for objects within the system. Not all objects require all three types of initialization.

The type `System_State` is an unsigned 8-bit integer definition. The package defines a set of constant declarations these constants act as masks corresponding to the various system states.

```
    Initialize  : constant := 2#0000_0001#;
    Normal      : constant := 2#0000_0010#;
    Shutdown    : constant := 2#0010_0000#;
```

4.3.4 Initialization_Manager

This package provides the services required to provide clients with the location (address) of their specific configuration data.

```
with Interfaces;
with System;
package Initialization_Manager is  --=========================================

    procedure Get_Configuration
               (The_Table        : in     Interfaces.Unsigned_32;
                Pre_Cert_Address :    out System.Address;
                Pre_Cert_Length  :    out Natural);

end Initialization_Manager; --================================================
```

Note: Goodrich provides the value of The_Table; It uniquely defines the third party developer's configuration data table in memory.

The users configuration data tables will usually manifest themselves internally as Ada record types, or in some cases arrays of Ada record types. During system initialization the software will “get” the configuration data by making calls to `Get_Configuration` that will return a parameter of type `System.Address`. This address value will be converted into an access or pointer type by using an instantiation of the predefined Ada package `Address_to_Access_Conversions`. By using this package and converting the address to an access type the user can achieve a “safe” overlay of an Ada record type over the storage occupied by the memory resident configuration data. For an example see package body Gonkulator 4.4.3.

The procedure `Get_Configuration` takes in a parameter named `The_Table` of type `Unsigned_32`. This parameter is a unique number that identifies the desired table that resides in system memory. The procedure also has two parameters that receive data, the parameter `Pre_Cert_Address` of type `System.Address` receives the physical memory address of the table in system memory, and the parameter `Pre_Cert_Length` of type `Natural` receives the length of the table in bytes.

```
    procedure Get_Configuration
               (The_Table        : in     Interfaces.Unsigned_32;
                Pre_Cert_Address :    out System.Address;
                Pre_Cert_Length  :    out Natural);
```

4.3.5 Package Data_Validity_Adt

This package provides the types that define all possible data validity attributes. This package also provides the routines required to set, check, and clear a validity object. Data_Validity data types are used exclusively in the construction of Data Repository items. Every Data Repository item has a component of type Data_Validity.

```
with Interfaces;
package Data_Validity_Adt is --================================================

    subtype Data_Validity is Interfaces.Unsigned_16;

    type Data_Validity_Attribute is
            (Valid,                    Invalid,
             No_Computed_Data,         Invalid_Computed_Data,
             Out_Of_Range_Hi,          Out_Of_Range_Lo,
             Invalid_Predecessor,      Defaulted,
             Defaulted_Used,           Old_Data,
             Rate_Error);

    type Validity_State is (On, Off);

                  --====================================--

   procedure Clear_All_Attributes (Item : in out Data_Validity);

                  ----------------------------------------

   procedure Set_Attribute (Item          : in out Data_Validity;
                            The_Attribute : in     Data_Validity_Attribute;
                            To_State      : in     Validity_State);

                  ----------------------------------------

   function Is_Invalid (Item : in Data_Validity) return Boolean;

                  ----------------------------------------

   function Valid return Data_Validity;

                  ----------------------------------------

   function Attribute_Is_Set (Item          : in Data_Validity;
                              The_Attribute : in Data_Validity_Attribute) return Boolean;

                  ----------------------------------------

    procedure Combine_Validity  (Item_1 : in  Data_Validity;
                                 Item_2 : in  Data_Validity;
                                 Result : out Data_Validity);

                  ----------------------------------------

    procedure Set_Default_Validity (Item : in out Data_Validity);

end Data_Validity_Adt; --=============================================================
```

4.3.5.1 Data_Validity_Adt Data Types

The type `Data_Validity` is a 32-bit integer type. It is essentially a private data type in that the actual bit fields that are set for any of the corresponding enumeration values of type `Data_Validity_Attribute` are hidden by the processing of the various subprograms defined in the package.

The type `Data_Validity_Attribute` is an enumeration type that contains the following values `Valid, Invalid, No_Computed_Data, Invalid_Computed_Data, Out_Of_Range_Hi, Out_Of_Range_Lo, Invalid_Predecessor, Defaulted, Defaulted_Used, Old_Data, Rate_Error`. The values describe various conditions and status that can be associated with a particular data repository item. Each value corresponds to a particular bit in the 32-bit data type `Data_Validity`. Several pairs of the values are mutually exclusive

4.3.5.2 Data_Validity Subprograms

*4.3.5.2.1 **procedure** Clear_All_Attributes*

This operation will clear all bits within the validity object.

```
procedure Clear_All_Attributes (Item : in out Data_Validity);
```

*4.3.5.2.2 **procedure** Set_Attribute*

This operation will set the appropriate attribute bit within the validity object. The caller of this operation supplies the attribute and the attribute state.

```
procedure Set_Attribute
        (Item          : in out Data_Validity;
         The_Attribute : in     Data_Validity_Attribute;
         To_State      : in     Validity_State);
```

*4.3.5.2.3 **function** Is_Invalid*

This operation will return the valid/invalid state of the validity object.

```
function Is_Invalid (Item : in Data_Validity) return Boolean;
```

*4.3.5.2.4 **function** Valid*

This operation will return the valid representation for a validity object.

```
function Valid return Data_Validity;
```

*4.3.5.2.5 **function** Attribute_Is_Set*

This operation will return the state of the specified attribute for the validity object.

```
function Attribute_Is_Set (Item          : in Data_Validity;
                           The_Attribute : in Data_Validity_Attribute) return Boolean;
```

*4.3.5.2.6 **procedure** Combine_Validity*

This operation will take two Validity objects and derive the resulting validity.

```
procedure Combine_Validity (Item_1 : in     Data_Validity;
                            Item_2 : in     Data_Validity;
                            Result :    out Data_Validity);
```

*4.3.5.2.7 **procedure** Set_Default_Validity*

The Set_Default_Validity operation will set the validity to defaulted.

```
procedure Set_Default_Validity (Item : in out Data_Validity);
```

4.3.6 Package Configuration_Id_Adt

This package provides the types that describe the configuration identifiers throughout the system. Included in this package are routines to determine the 'class' and 'instance' numbers that are embedded in the configuration identifier.

```
with Interfaces;

package Configuration_Id_Adt is -- ========================================

    subtype Configuration_Id is Interfaces.Unsigned_32;
    subtype Class_Id         is Interfaces.Unsigned_16;
    subtype Instance_Id      is Interfaces.Unsigned_16;
    --
    -- The following subtypes identify the different classes.  The range of the
    -- subtype specified the number of instances allowed for the subtype.

    subtype Data_Set_Id            is Configuration_Id range 16#0700_0000# .. 16#0700_0032#; -- 50
    subtype Repository_Id          is Configuration_Id range 16#0D00_0000# .. 16#0D00_05DC#;
    subtype Event_Id               is Configuration_Id range 16#0F00_0000# .. 16#0F00_01F4#;
    subtype Monitored_Event_Id     is Configuration_Id range 16#0F00_0001# .. 16#0F00_012C#;
    subtype Instantaneous_Event_Id is Configuration_Id range 16#0F00_012D# .. 16#0F00_01F4#;
    subtype Threshold_Id           is Configuration_Id range 16#0F10_0000# .. 16#0F10_05DC#;
    subtype Event_Data_Set_Id      is Configuration_Id range 16#0FB0_0000# .. 16#0FB0_0032#; -- 50


    Nil_Id : constant Configuration_Id := 16#0000_0000#;

                  --------------------------------------

   function  Class_Is       (The_Id : in     Configuration_Id) return Class_Id;

   function  Instance_Is    (The_Id : in     Configuration_Id) return Instance_Id;

   function  Nil_Identifier (The_Id : in     Configuration_Id) return Boolean;

end Configuration_Id_Adt; --=================================================
```

4.3.6.1 Package Configuration_ID_Adt Data Types

The package defines three main subtypes; they are `Configuration_Id`, `Class_Id`, and `Instance_Id`. The `Configuration_ID` subtype is a 32 bit unsigned integer type that is used for unique identifiers that identify configuration data table that reside in memory. The unique identifier that Goodrich supplies the third party vendors to access their configuration data is of this subtype. Each value of type `Configuration_Id` can be decomposed into two 16-bit halves, these two parts correspond to values of the subtypes `Class_Id`, and `Instance_Id`.

```
    subtype Configuration_Id is Interfaces.Unsigned_32;
    subtype Class_Id         is Interfaces.Unsigned_16;
    subtype Instance_Id      is Interfaces.Unsigned_16;
```

The package also contains a number of subtype ranges each corresponding to a particular range of Id numbers that identify configuration data for various type of system objects.

```
subtype Data_Set_Id            is Configuration_Id range 16#0700_0000# .. 16#0700_0032#; -- 50
subtype Repository_Id          is Configuration_Id range 16#0D00_0000# .. 16#0D00_05DC#;
subtype Event_Id               is Configuration_Id range 16#0F00_0000# .. 16#0F00_01F4#;
subtype Monitored_Event_Id     is Configuration_Id range 16#0F00_0001# .. 16#0F00_012C#;
subtype Instantaneous_Event_Id is Configuration_Id range 16#0F00_012D# .. 16#0F00_01F4#;
subtype Threshold_Id           is Configuration_Id range 16#0F10_0000# .. 16#0F10_05DC#;
subtype Event_Data_Set_Id      is Configuration_Id range 16#0FB0_0000# .. 16#0FB0_0032#; -- 50
Nil_Id : constant Configuration_Id := 16#0000_0000#;
```

4.3.6.2 Package Configuration_ID_Adt Subprograms

*4.3.6.2.1 **function** Class_Is*

This operation will return the class number from the specified configuration identifier.

```
function Class_Is (The_Id : in Configuration_Id) return Class_Id;
```

*4.3.6.2.2 **function** Instance_Is*

This operation will return the instance number from the specified configuration identifier.

```
function  Instance_Is (The_Id : in Configuration_Id) return Instance_Id;
```

*4.3.6.2.3 **function** Nil_Identifier*

This operation will determine if the Configuration Id represents a Nil identifier (i.e. The Instance id = 0);

```
function Nil_Identifier (The_Id : in Configuration_Id) return Boolean;
```

4.3.7 Package Cc_Executive_Interface

The Executive_Interface provides the interface methods necessary for a class to be scheduled by the executive. Each class that is to be controlled by the executive scheduler must inherit this interface.

```
with System_States;

use type System_States.Operation_States;

package Cc_Executive_Interface is  --=======================================

    type Object    is abstract tagged limited private;

    type Reference is access all      Object'Class;
    type View      is access constant Object'Class;


    procedure Update   (The_Object : access Object) is abstract;

    procedure Shutdown (The_Object : access Object) is abstract;

private  --=========================================================================

  type Object is abstract tagged limited
    record
     Registered_For_Shutdown            : Boolean :=False;
     Registered_For_Change_Of_Operation : Boolean := False;
     Operation_State                    : System_States.Operation_States :=
                                          System_States.Operation_In_Progress;
    end record;

end Cc_Executive_Interface;  --=================================================
```

4.3.7.1 Package CC_Executive_Interface Data Types

For an explanation of the types Object, Reference, and View see 4.3.1.

The private part of the package is supplied to show the components that are defined as part of the tagged record type OBJECT.

The type is a tagged record type which means that it can be extended by the user who derives a new type from it. The components of the record are revealed in the Software Interface so that a user does not extend the type and use the names of any of the existing components.

```
 type Object is abstract tagged limited
   record
     Registered_For_Shutdown            : Boolean :=False;
     Registered_For_Change_Of_Operation : Boolean := False;
     Operation_State                    : System_States.Operation_States :=
                                          System_States.Operation_In_Progress;
   end record;
```

4.3.7.2 Package CC_Executive_Interface Subprograms

*4.3.7.2.1 **procedure** Update*

The Update operation defines the interface specification to the Update method supplied by each class that inherits this interface. The Update method supplied by the class will allow the scheduler to initiate any periodic processing required for the classes objects.

```
    procedure Update (The_Object : access Object) is abstract;
```

*4.3.7.2.2 **procedure** Shutdown*

The Shutdown procedure supplied by the class performs the processing required to prepare the classes' object for a system shutdown. The shutdown procedure does not release memory that has been allocated to the object.

A call to this operation will result when the system has been notified of an impending shutdown. This does not guarantee that a shutdown will actually occur. Normal processing may resume after this call is made.

```
    procedure Shutdown (The_Object : access Object) is abstract;
```

4.3.8 Package Fc_Executive_Interface

The FC_Executive_Interface provides the interface methods necessary for each factory class in the system. Factories are usually responsible for the creation of all of the objects of a particular class. As well as creating the objects of a particular class it will usually provide a specific function to dispense references to the objects it creates internally. A factory class will also have to initially register with the Executive; this is because the executive is responsible for calling the Initialize procedure during the various phases of initialization.

```
with System_States;

package Fc_Executive_Interface is -- ========================================


    type Object    is abstract tagged limited private;

    type Reference is access all      Object'Class;
    type View      is access constant Object'Class;

                   --===================================--

    procedure Initialize
            (The_Factory : access Object;
             The_Mode    : in     System_States.Initialization_Mode) is abstract;

Private  --=========================================================================

    type Object is abstract tagged limited null record;

end Fc_Executive_Interface; --================================================
```

4.3.8.1 Package FC_Executive_Interface Type Declarations

```
type Object    is abstract tagged limited private;

type Reference is access all      Object'Class;
type View      is access constant Object'Class;
```

For an explanation of the types `Object, Reference`, and `View` see section 4.3.1.

4.3.8.2 Package FC_Executive_Interface Subprograms

*4.3.8.2.1 **procedure** Initialize*

The Initialize operation defines the interface specification to the Initialize method supplied by each class that inherits this interface. The Initialize method supplied by the factory class shall perform the initialization required to create all objects managed by the factory, and then resolve all run-time references between the objects created and other objects in the system.

The Executive will call this operation during system initialization.

When this operation is called with the mode set to 'Create_Objects', the factory class should create all objects that will be managed by that factory. When this Operation is called with the mode set to 'Resolve_References', the routine will call Resolve Reference routines for all of the objects it manages.

```
procedure Initialize (The_Factory : access Object;
                      The_Mode    : in     System_States.Initialization_Mode) is abstract;
```

4.3.9 Package Executive

The Executive package provides the tools for controlling the scheduling of operations within the system. The main function of the Executive package is to provide the capability to periodically schedule various system objects defined within the system.

```
with Cc_Executive_Interface;
with Configuration_Id_Adt;
with Fc_Executive_Interface;
with Interfaces;
with System_States;
with System_Types;

package Executive is --=====================================================


    type Startup_Mode is (Warm_Start, Cold_Start);

    type Execution_Rate_Id is new Integer range 0 .. 6;
    for Execution_Rate_Id'Size use 8;

                --====================================--

    function Startup_Mode_Is return Startup_Mode;

    function Rate_In_Milliseconds (Rate_Id : in Execution_Rate_Id) return Interfaces.Unsigned_32;

                ----------------------------------------

    procedure Register
            (The_Object     : in out Cc_Executive_Interface.Reference;
             With_Config_Id : in     Configuration_Id_Adt.Configuration_Id;
             The_Status     :    out System_Types.Return_Status);

                ----------------------------------------

    procedure Register
            (The_Factory : in out Fc_Executive_Interface.Reference);


end Executive; --=============================================================
```

4.3.9.1 Package Executive Type Declarations

```
    type Startup_Mode is (Warm_Start, Cold_Start);

    type Execution_Rate_Id is new Integer range 0 .. 6;
    for Execution_Rate_Id'Size use 8;
```

The type `Startup_Mode` is an enumeration type that enumerates the values cooresponding to the different types of startups the system can experience. The function `Startup_Mod_Is` return a value of this type.

The type `Executation_Rate_ID` is an integer type that defines the total number of different scheduling rates that are available within the system. The function `Rate_In_Milliseconds` takes this type as a parameter and returns the number of milliseconds defined for that rates period.

4.3.9.2 Package Executive Subprograms

*4.3.9.2.1 **function** Startup_Mode_Is*

This operation will return the startup mode for the system.

```
function Startup_Mode_Is return Startup_Mode;
```

*4.3.9.2.2 **function** Rate_In_Milliseconds*

This operation will return the number of milliseconds related to the specified periodic execution rate.

```
function Rate_In_Milliseconds (Rate_Id : in Execution_Rate_Id) return Interfaces.Unsigned_32;
```

*4.3.9.2.3 **procedure** Register {1}*

The Register operation is provided by this interface to allow executive clients to register with the executive. The executive will be provided a set of configuration data during its initialization. This configuration data will identify each of the objects in the system that will be scheduled by the executive, and its rate of execution. At run-time, all objects to be scheduled must register using this service to provide the run-time binding between the executive and each scheduable client.

```
procedure Register
        (The_Object     : in out Cc_Executive_Interface.Reference;
         With_Config_Id : in     Configuration_Id_Adt.Configuration_Id;
         The_Status     :    out System_Types.Return_Status);
```

*4.3.9.2.4 **procedure** Register {2}*

The Register operation is provided to allow executive factory clients to register with the executive. The clients that register will then be called using the operations supplied by the Fc_Executive_Interface interface, to initialize the system.

```
procedure Register (The_Factory : in out Fc_Executive_Interface.Reference);
```

4.3.10 Package Timestamp_Adt

This package provides the types that contain the timestamp and relative timestamp (time since baseline) used throughout the system.

```
with Ada.Real_Time;
with Cc_Executive_Interface;
with Interfaces;

use type Interfaces.Unsigned_32;

package Timestamp_Adt is -- ================================================


    type Object    is new Cc_Executive_Interface.Object with private;
    type Reference is access all Object'Class;
    type View      is access constant Object'Class;


    type Absolute_Timestamp is
        record
            Year              : String (1 .. 4);          -- YYYY - current UTC year
            Month             : String (1 .. 2);          -- MM   - current UTC month
            Day               : String (1 .. 2);          -- DD   - current UTC day
            Time_Marker       : String (1 .. 1) := "T";   -- T    - literal 'T'
            Hour              : String (1 .. 2);          -- HH   - current UTC hour
            Minute            : String (1 .. 2);          -- MM   - current UTC minute
            Whole_Second      : String (1 .. 2);          -- SS   - current UTC second
            Second_Marker     : String (1 .. 1) := ",";   -- ,    - literal ','
            Fractional_Second : String (1 .. 3);          -- sss  - fractional seconds
            Utc_Marker        : String (1 .. 1) := "Z";   -- Z    - literal 'Z'
        end record;


    type Relative_Timestamp is private;

    Nil_Relative_Timestamp : constant Relative_Timestamp;

    Default_Absolute_Timestamp : constant Absolute_Timestamp :=
                    (Year              => "1998",
                     Month             => "01",
                     Day               => "01",
                     Time_Marker       => "T",
                     Hour              => "00",
                     Minute            => "00",
                     Whole_Second      => "00",
                     Second_Marker     => ",",
                     Fractional_Second => "000",
                     Utc_Marker        => "Z");


                   --====================================--


    function Absolute_Timestamp_Is           return Absolute_Timestamp;

    function Absolute_Reference_Timestamp_Is return Absolute_Timestamp;

    function Relative_Timestamp_Is           return Relative_Timestamp;
```

```
function Relative_Timestamp_Is  (Start_Time : in Relative_Timestamp;
                                 Delta_Time : in Duration) return Relative_Timestamp;

function Previous_Relative_Timestamp_Is (Start_Time : Relative_Timestamp;
                                         Delta_Time : Duration) return Relative_Timestamp;

function In_Range (The_Timestamp  : in  Relative_Timestamp;
                   The_Start_Time : in  Relative_Timestamp;
                   The_End_Time   : in  Relative_Timestamp) return Boolean;

function Elapsed_Ms (The_Start_Time : in  Relative_Timestamp;
                     The_End_Time   : in  Relative_Timestamp) return Relative_Timestamp;

function Elapsed_Ms (The_Start_Time : in  Relative_Timestamp;
                     The_End_Time   : in  Relative_Timestamp) return Interfaces.Unsigned_32;

function Previous_Time_Is (The_Delta : in Duration) return Relative_Timestamp;

procedure Add (The_Accumulator : in out Relative_Timestamp;
               The_Delta       : in     Relative_Timestamp);


procedure Update   (The_Object : access Object);

procedure Shutdown (The_Object : access Object);

function Relative_To_Absolute_Time (The_Relative_Time : in Relative_Timestamp)
                                    return Absolute_Timestamp;

function Relative_To_String (The_Relative_Time : in Relative_Timestamp)
                             return String;

function "="  (Left  : in  Absolute_Timestamp;
               Right : in  Absolute_Timestamp) return Boolean;

function "<"  (Left  : in  Absolute_Timestamp;
               Right : in  Absolute_Timestamp) return Boolean;

function "<=" (Left  : in  Absolute_Timestamp;
               Right : in  Absolute_Timestamp) return Boolean;

function ">"  (Left  : in  Absolute_Timestamp;
               Right : in  Absolute_Timestamp) return Boolean;

function ">=" (Left  : in  Absolute_Timestamp;
               Right : in  Absolute_Timestamp) return Boolean;

function "="  (Left  : in  Relative_Timestamp;
               Right : in  Relative_Timestamp) return Boolean;

function "<"  (Left  : in  Relative_Timestamp;
               Right : in  Relative_Timestamp) return Boolean;

function "<=" (Left  : in  Relative_Timestamp;
               Right : in  Relative_Timestamp) return Boolean;

function ">"  (Left  : in  Relative_Timestamp;
               Right : in  Relative_Timestamp) return Boolean;


function ">=" (Left  : in  Relative_Timestamp;
               Right : in  Relative_Timestamp) return Boolean;
```

```
    function "+"  (Left  : in  Relative_Timestamp;
                   Right : in  Relative_Timestamp) return Relative_Timestamp;

private

    -- Represents the number of milliseconds from reference point
    type Relative_Timestamp is new Interfaces.Unsigned_32;

    type Object is new Cc_Executive_Interface.Object
        with record
            Current_Relative_Time           : Relative_Timestamp;
            Absolute_Time_Reference         : Ada.Real_Time.Time;
            Absolute_Time_Reference_Iso8601 : Absolute_Timestamp;
        end record;

    Nil_Relative_Timestamp : constant Relative_Timestamp := 0;

end Timestamp_Adt;
```

4.3.10.1 Package Timestamp_Adt Type Declarations

```
    type Object is new Cc_Executive_Interface.Object with private;

    type Reference is access all Object'Class;

    type View is access constant Object'Class;
```

The type `Object` is derived from the CC_Executive_Interface.Object and is extended in the private part of this package. The reason for this object is so the package itself can be 'scheduled' in order to keep track of time. Although these types are necessary for the package to work properly, they are of no concern to the end user. The real working type of this package is the Relative_Timestamp Absolute_Timestamp.

```
    type Absolute_Timestamp is
        record
            Year              : String (1 .. 4);
            Month             : String (1 .. 2);
            Day               : String (1 .. 2);
            Time_Marker       : String (1 .. 1) := "T";
            Hour              : String (1 .. 2);
            Minute            : String (1 .. 2);
            Whole_Second      : String (1 .. 2);
            Second_Marker     : String (1 .. 1) := ",";
            Fractional_Second : String (1 .. 3);
            Utc_Marker        : String (1 .. 1) := "Z";
        end record;

    type Relative_Timestamp is private;

    Nil_Relative_Timestamp : constant Relative_Timestamp;

    Default_Absolute_Timestamp : constant Absolute_Timestamp :=
                    (Year              => "1998",  Month             => "01",
                     Day               => "01",    Time_Marker       => "T",
                     Hour              => "00",    Minute            => "00",
                     Whole_Second      => "00",    Second_Marker     => ",",
                     Fractional_Second => "000",   Utc_Marker        => "Z");
```

The Absolute and Relative timestamp types are ones of real interest to the end user. The absolute timestamp is an exposed record type with a set of components that are broken up into string values. The format of the string conforms

to ISO 8601. Within HUMS, Absolute timestamps are used to establish a time base and the Relative timestamps are used after that. As can be seen Absolute require a considerable amount of storage compared to the 32 bit unsigned integer values of Relative timestamps.

Keep in mind that Ada supplies a package named Ada.Real_Time that supplies an extremely efficient and accurate concept of time, that package is described in detail in the Ada Language Reference Manual section D.8.

4.3.10.2 Package Timestamp_Adt Subprograms

*4.3.10.2.1 **function** Absolute_Timestamp_Is*

This operation will return a timestamp representing the current UTC time in ISO 8601 format. This format is:

YYYYMMDDTHHMMSS,sssZ

where:

YYYY	- current UTC year
MM	- current UTC month
DD	- current UTC day
T	- literal 'T'
HH	- current UTC hour
MM	- current UTC minute
SS	- current UTC second
,	- literal ','
sss	- fractional seconds
Z	- literal 'Z' (indicates time is in UTC or Zulu)

```
function Absolute_Timestamp_Is return Absolute_Timestamp;
```

*4.3.10.2.2 **function** Absolute_Reference_Timestamp_Is*

This operation will return a timestamp representing the reference time (absolute time format) from which all relative timestamps are based.

```
function Absolute_Reference_Timestamp_Is return Absolute_Timestamp;
```

*4.3.10.2.3 **function** Relative_Timestamp_Is*

This operation will return a relative timestamp.

```
function Relative_Timestamp_Is return Relative_Timestamp;
```

*4.3.10.2.4 **function** Relative_Timestamp_Is*

This operation will return a relative timestamp.

```
function Relative_Timestamp_Is (Start_Time : in Relative_Timestamp;
                                Delta_Time : in Duration) return Relative_Timestamp;
```

*4.3.10.2.5 **function** Previous_Relative_Timestamp_Is*

This operation will return a relative timestamp, which is before the supplied start time. Note the relative timestamp will be limited to zero.

```
function Previous_Relative_Timestamp_Is (Start_Time : Relative_Timestamp;
                                         Delta_Time : Duration) return Relative_Timestamp;
```

*4.3.10.2.6 **function** In_Range*

This operation will return an indication of whether a relative timestamp is within to other relative timestamps.

```
function In_Range (The_Timestamp  : in Relative_Timestamp;
                   The_Start_Time : in Relative_Timestamp;
                   The_End_Time   : in Relative_Timestamp) return Boolean;
```

*4.3.10.2.7 **function** Elapsed_Ms {1}*

This operation will return the elapsed time, in milliseconds, between two relative timestamps.

```
function Elapsed_Ms
        (The_Start_Time : in     Relative_Timestamp;
         The_End_Time   : in     Relative_Timestamp) return Relative_Timestamp;
```

*4.3.10.2.8 **function** Elapsed_Ms {2}*

This operation will return the elapsed time, in milliseconds, between two relative timestamps.

```
function Elapsed_Ms (The_Start_Time : in Relative_Timestamp;
                     The_End_Time   : in Relative_Timestamp) return Interfaces.Unsigned_32;
```

*4.3.10.2.9 **function** Previous_Time_Is*

This operation will return the current time minus the supplied delta. It this previous time is less than zero, it will be set to zero.

```
function Previous_Time_Is (The_Delta : in Duration) return Relative_Timestamp;
```

*4.3.10.2.10 **procedure** Add*

This operation will return the sum of two relative timestamps.

```
procedure Add (The_Accumulator : in out Relative_Timestamp;
               The_Delta       : in     Relative_Timestamp);
```

*4.3.10.2.11 **procedure** Update*

This operation will update the current and relative time attributes based upon the real time clock (RTC). This operation should be called at the highest frequency of updates within the system. This procedure allows the package to keep track of the current time on a cyclic basis. The user of this package need not be concerned with it.

```
procedure Update (The_Object : access Object);
```

*4.3.10.2.12 **procedure** Shutdown*

This operation is required due to the inheritance of the executive interface, however no shutdown processing is required for this class.

```
procedure Shutdown (The_Object : access Object);
```

*4.3.10.2.13 **function** Relative_To_Absolute_Time*

This operation converts relative time to absolute time.

```
function Relative_To_Absolute_Time (The_Relative_Time : in Relative_Timestamp) return
                                    Absolute_Timestamp;
```

*4.3.10.2.14 **function** Relative_To_String*

This operation converts relative time to formatted string (hh:mm:ss).

```
function Relative_To_String (The_Relative_Time : in  Relative_Timestamp) return String;
```

4.3.10.2.15 Overloaded Relational Operators for Absolute_Timestamp

The following are a group of overloaded relational operators, they each take two parameters of Absolute_Timestamp and all return values of type Boolean. The functions perform their general relational operations with no surprises.

```
function "="  (Left  : in Absolute_Timestamp;
               Right : in Absolute_Timestamp) return Boolean;

function "<"  (Left  : in Absolute_Timestamp;
               Right : in Absolute_Timestamp) return Boolean;

function "<=" (Left  : in Absolute_Timestamp;
               Right : in Absolute_Timestamp) return Boolean;

function ">"  (Left  : in Absolute_Timestamp;
               Right : in Absolute_Timestamp) return Boolean;

function ">=" (Left  : in Absolute_Timestamp;
               Right : in Absolute_Timestamp) return Boolean;
```

4.3.10.2.16 Overloaded relational Operators for Relative_Timestamp

The following are a group of overloaded relational operators, they each take two parameters of Relative_Timestamp and all return values of type Boolean. The functions perform their general relational operations with no surprises.

```
    function "="  (Left  : in Relative_Timestamp;
                   Right : in Relative_Timestamp) return Boolean;


    function "<"  (Left  : in Relative_Timestamp;
                   Right : in Relative_Timestamp) return Boolean;


    function "<=" (Left  : in Relative_Timestamp;
                   Right : in Relative_Timestamp) return Boolean;


    function ">"  (Left  : in Relative_Timestamp;
                   Right : in Relative_Timestamp) return Boolean;


    function ">=" (Left  : in Relative_Timestamp;
                   Right : in Relative_Timestamp) return Boolean;
```

*4.3.10.2.17 **function** "+"*

This function simply adds to relative time values together and returns a Relative_timestamp.

```
    function "+" (Left  : in     Relative_Timestamp;
                  Right : in     Relative_Timestamp) return Relative_Timestamp;
```

4.3.11 package Data_Item

This class defines the base type of data stored in the data repository. Each data item store in the repository will contain the data defined in this class. This class only provides the common data elements required for all types of data in the repository. This class must be extended for each specific data type stored in the repository.

```
with Data_Validity_Adt;
with Timestamp_Adt;

package Data_Item is -- ===================================================

    type Object is abstract tagged
        record
            Validity  : Data_Validity_Adt.Data_Validity  := Data_Validity_Adt.Valid;
            Timestamp : Timestamp_Adt.Relative_Timestamp := Timestamp_Adt.Nil_Relative_Timestamp;
        end record;

    type Reference is access all Object'Class;

    type View is access constant Object'Class;

    procedure Set_Defaults (The_Item : in out Object);

end Data_Item; -- ============================================================
```

4.3.11.1 Package Data_Item Type Declarations

This packages `Object` type is not private but exposed so that it can be derived without have to become part of the packages family.

The tagged record type contains two components that all derived objects will contain, a `Validity` component which indicates the validity of the data item, and a `Timetamp` component which will represent a relative offset from a known reference within the system using the repository. These components are of types that are described in there corresponding packages.

```
type Object is abstract tagged
   record
     Validity  : Data_Validity_Adt.Data_Validity := Data_Validity_Adt.Valid;
     Timestamp : Timestamp_Adt.Relative_Timestamp := Timestamp_Adt.Nil_Relative_Timestamp;
   end record;

type Reference is access all Object'Class;

type View is access constant Object'Class;
```

4.3.11.2 Package Data_Item Subprograms

4.3.11.2.1 ***procedure*** *Set_Defaults*

This procedure is used to allow each new defined data type to have a procedure called upon the Object at creation time.

```
procedure Set_Defaults (The_Item : in out Object);
```

4.3.12 package Repository_Types

This package contains any specific types required for the repository CSC. This is a simple package that only defines a String subtype constrained to 20 characters and constant of that type filled with blanks.

```
package Repository_Types is --=================================================

    subtype String_20 is String (1 .. 20);

    Blank_String_20 : constant String_20 := (others => ' ');

end Repository_Types; --=======================================================
```

4.3.13 package Abstract_Repository

This package contains the abstract class type for all repositories. All repository types must inherit this interface to allow a common base class type to be used when accessing any of the derived repository classes. There are no operations defined for this abstract class.

```
package Abstract_Repository is --=============================================

    type Object    is abstract tagged private;

    type Reference is access all Object'Class;

    type View      is access constant Object'Class;

private --=====================================================================

    type Object is abstract tagged null record;

end Abstract_Repository; --====================================================
```

Note : For an explanation of the types `Object, Reference`, and `View` see section 4.3.1.

4.3.14 package Generic_Repository

This package contains the generic class that maintains the 'repository' for individual data items. The package supports requests for the single current value of a data item, multiple values of a data item, or a set of time based values for a data item. The amount of historical (or queued) data retained within the repository instance is specified via a generic parameter during creation of the object.

This package inherits the base repository class. It also inherits the abstract data item type for the data contained in each item. The type specified in the generic instantiation of this package extends this type.

```
with Abstract_Repository;
with Data_Item;
with System;
with Timestamp_Adt;

generic --=====================================================================

    type Data_Type is private;
    Default_Value           : in     Data_Type;
    Max_Data_Elements       : in     Integer;
    Special_Pool_Enabled    : in     Boolean;
    Special_Pool_Address    : in out System.Address;
    Special_Pool_Data_Valid : in     Boolean;
    Max_Sp_Data_Elements    : in     Integer;

package Generic_Repository is --===============================================

    type Data_Element is limited private;

    type Data is new Data_Item.Object
        with record
            Value : Data_Type;
        end record;

                --===================================--
```

```
    type Object    is new Abstract_Repository.Object with private;

    type Reference is access all      Object'Class;
    type View      is access constant Object'Class;

    type Item_Buffer is array (Integer range <>) of Data;

                --------------------------------------

    function Number_Of_Elements
            (The_Repository  : in     View) return Integer;

    function Number_Of_Elements
            (The_Repository  : in     View;
             Start_Time      : in     Timestamp_Adt.Relative_Timestamp;
             End_Time        : in     Timestamp_Adt.Relative_Timestamp :=
             Timestamp_Adt.Nil_Relative_Timestamp) return Integer;

    function Get_Current
            (From_Repository : in     View) return Data;

    function Get_Previous
            (From_Repository : in     View;
             Prev_Sample_Num : in     Integer) return Data;

    procedure Get_Buffer
            (From_Repository : in     View;
             Number_Of_Items : in out Integer;
             The_Buffer      :    out Item_Buffer);

    procedure Get_Buffer
            (From_Repository : in     View;
             Start_Time      : in     Timestamp_Adt.Relative_Timestamp;
             End_Time        : in     Timestamp_Adt.Relative_Timestamp :=
                                      Timestamp_Adt.Nil_Relative_Timestamp;
             The_Buffer      :    out Item_Buffer;
             Number_Of_Items :    out Integer);

    procedure Put
            (At_Repository   : in out Reference;
             The_Item        : in     Data);


private --=========================================================================


    type Data_Element is
        record
            Dummy : Boolean;
        end record;

    type Object is new Abstract_Repository.Object with null record;

end Generic_Repository; --=====================================================
```

4.3.14.1 Package Generic_Repository Generic Parameter Declarations

The type Data_Type is a private generic type parameter that is imported into the generic unit. It is used to add components to internal data types through record extensions.

```
type Data_Type is private;
```

4.3.14.2 Package Generic_Repository Type Declarations

Extend the data item type with the 'Data_Type' specified during generic instantiation. All data types maintained in the repository are based upon the abstract type specified in the 'Data_Item' class.

```
type Data_Element is limited private;

type Data is new Data_Item.Object
    with record
        Value : Data_Type;
    end record;
```

The repository 'object' **is** a specialization of the abstract repository class.

```
type Object    is new Abstract_Repository.Object with private;

type Reference is access all Object'Class;
type View      is access constant Object'Class;
```

The item queue contains the multiple values of a data item.

```
type Item_Buffer is array (Integer range <>) of Data;
```

4.3.14.3 Package Generic_Repository Subprograms

*4.3.14.3.1 **function** Number_Of_Elements {1}*

The Number_Of_Elements operation will return the total number of data values in the repository for the data item.

```
function Number_Of_Elements (The_Repository : in View) return Integer;
```

*4.3.14.3.2 **function** Number_Of_Elements {2}*

The Number_Of_Elements operation will return the number of data values in the repository for the data item with timestamps within the start and end times specified.

```
function Number_Of_Elements
        (The_Repository : in  View;
         Start_Time     : in  Timestamp_Adt.Relative_Timestamp;
         End_Time       : in  Timestamp_Adt.Relative_Timestamp :=
                              Timestamp_Adt.Nil_Relative_Timestamp)
         return Integer;
```

*4.3.14.3.3 **function** Get_Current*

The Get operation will return the most current data for the data item.

```
function Get_Current (From_Repository : in View) return Data;
```

*4.3.14.3.4 **function** Get_Previous*

The Get operation will return the specified previous sample for the data item. If that sample does not yet exist in the buffer, the oldest item in the buffer will be returned.

```
function Get_Previous (From_Repository : in View;
                       Prev_Sample_Num : in Integer) return Data;
```

*4.3.14.3.5 **procedure** Get_Buffer {1}*

The Get_Buffer operation will return a buffer containing the data for the specified data item. The actual number of items placed in the buffer is returned to the caller along with the buffer. The output buffer is filled starting with the most recent item in the repository.

```
procedure Get_Buffer (From_Repository : in     View;
                      Number_Of_Items : in out Integer;
                      The_Buffer      :    out Item_Buffer);
```

*4.3.14.3.6 **procedure** Get_Buffer {2}*

The Get Buffer operation will return a buffer containing the data for the specified data item for the requested period of time. The specified end time must be no later than the current time. The actual number of items placed in the buffer is returned to the caller along with the buffer. The output buffer is filled starting with the most recent item in the repository.

If no end time is specified, the current time will be used.

```
procedure Get_Buffer
        (From_Repository : in     View;
         Start_Time      : in     Timestamp_Adt.Relative_Timestamp;
         End_Time        : in     Timestamp_Adt.Relative_Timestamp :=
                                  Timestamp_Adt.Nil_Relative_Timestamp;
         The_Buffer      :    out Item_Buffer;
         Number_Of_Items :    out Integer);
```

*4.3.14.3.7 **procedure** Put*

The Put operation allows a producer client to update the current data for a data item in the repository.

```
 procedure Put (At_Repository : in out Reference;
                The_Item      : in     Data);
```

4.3.15 package Repository

```
with Abstract_Repository;
with Configuration_Id_Adt;
with Data_Item;
with Fc_Executive_Interface;
with Generic_Repository;
with Interfaces;
with Repository_Types;
with System_States;
with Timestamp_Adt;
with System;
with System.Storage_Elements;
with System_Types;

use  System.Storage_Elements;       -- for address addition

package Repository is ------------------------------------------------------

    Max_Fp_Items             : constant Natural := 1000;
    Max_Int_Items            : constant Natural := 200;
    Max_Discrete_Items       : constant Natural := 300;
    Max_Unsigned_Items       : constant Natural := 175;
    Max_String_Items         : constant Natural := 40;
    Max_Abs_Time_Items       : constant Natural := 20;
    Max_Rel_Time_Items       : constant Natural := 20;

    Max_Fp_Elements          : constant Integer := 15_000;
    Max_Int_Elements         : constant Integer := 1_500;
    Max_Dis_Elements         : constant Integer := 2_000;
    Max_Unsigned_Elements    : constant Integer := 400;
    Max_String_Elements      : constant Integer := 200;
    Max_Abs_Time_Elements    : constant Integer := 20;
    Max_Rel_Time_Elements    : constant Integer := 20;

    Max_Sp_Fp_Items          : constant Natural := 200;
    Max_Sp_Int_Items         : constant Natural := 100;
    Max_Sp_Discrete_Items    : constant Natural := 50;
    Max_Sp_Unsigned_Items    : constant Natural := 200;
    Max_Sp_String_Items      : constant Natural := 40;
    Max_Sp_Abs_Time_Items    : constant Natural := 10;
    Max_Sp_Rel_Time_Items    : constant Natural := 10;

    Max_Sp_Fp_Elements       : constant Integer := Max_Sp_Fp_Items;
    Max_Sp_Int_Elements      : constant Integer := Max_Sp_Int_Items;
    Max_Sp_Dis_Elements      : constant Integer := Max_Sp_Discrete_Items;
    Max_Sp_Unsigned_Elements : constant Integer := Max_Sp_Unsigned_Items;
    Max_Sp_String_Elements   : constant Integer := Max_Sp_String_Items;
    Max_Sp_Abs_Time_Elements : constant Integer := Max_Sp_Abs_Time_Items;
    Max_Sp_Rel_Time_Elements : constant Integer := Max_Sp_Rel_Time_Items;

    Sp_Start_Address : System.Address;

    Sp_Data_Valid    : constant Boolean := True;
```

```
    -- ======================================================================

    type Object    is new Fc_Executive_Interface.Object with private;

    type Reference is access all Object'Class;

    type View      is access constant Object'Class;

    subtype Data_Array is System_Types.Byte_Array (1 .. 4);

    type Data is new Data_Item.Object
        with record
            Value : Data_Array;
        end record;

    type Item_Buffer is array (Integer range <>) of Data;

                  ---------------------------------------

    package Fp_Repository is new Generic_Repository
                (Data_Type               => Interfaces.Ieee_Float_32,
                 Default_Value           => 0.0,
                 Max_Data_Elements       => Max_Fp_Elements,
                 Special_Pool_Enabled    => True,
                 Special_Pool_Address    => Sp_Start_Address,
                 Special_Pool_Data_Valid => Sp_Data_Valid,
                 Max_Sp_Data_Elements    => Max_Sp_Fp_Elements);

                  ---------------------------------------

package Int_Repository is new Generic_Repository
                (Data_Type               => Interfaces.Integer_32,
                 Default_Value           => 0,
                 Max_Data_Elements       => Max_Int_Elements,
                 Special_Pool_Enabled    => True,
                 Special_Pool_Address    => Sp_Start_Address,
                 Special_Pool_Data_Valid => Sp_Data_Valid,
                 Max_Sp_Data_Elements    => Max_Sp_Int_Elements);

                  ---------------------------------------

package Discrete_Repository is new Generic_Repository
                (Data_Type               => Boolean,
                 Default_Value           => False,
                 Max_Data_Elements       => Max_Dis_Elements,
                 Special_Pool_Enabled    => True,
                 Special_Pool_Address    => Sp_Start_Address,
                 Special_Pool_Data_Valid => Sp_Data_Valid,
                 Max_Sp_Data_Elements    => Max_Sp_Dis_Elements);

                  ---------------------------------------

package Unsigned_Repository is new Generic_Repository
                (Data_Type               => Interfaces.Unsigned_32,
                 Default_Value           => 0,
                 Max_Data_Elements       => Max_Unsigned_Elements,
                 Special_Pool_Enabled    => True,
                 Special_Pool_Address    => Sp_Start_Address,
                 Special_Pool_Data_Valid => Sp_Data_Valid,
                 Max_Sp_Data_Elements    => Max_Sp_Unsigned_Elements);

                  ---------------------------------------
```

```
package String_Repository is new Generic_Repository
               (Data_Type              => Repository_Types.String_20,
                Default_Value          => (others => ' '),
                Max_Data_Elements      => Max_String_Elements,
                Special_Pool_Enabled   => True,
                Special_Pool_Address   => Sp_Start_Address,
                Special_Pool_Data_Valid => Sp_Data_Valid,
                Max_Sp_Data_Elements   => Max_Sp_String_Elements);

                  --====================================--

package Absolute_Timestamp_Repository is new Generic_Repository
               (Data_Type              => Timestamp_Adt.Absolute_Timestamp,
                Default_Value          => Timestamp_Adt.Default_Absolute_Timestamp,
                Max_Data_Elements      => Max_Abs_Time_Elements,
                Special_Pool_Enabled   => True,
                Special_Pool_Address   => Sp_Start_Address,
                Special_Pool_Data_Valid => Sp_Data_Valid,
                Max_Sp_Data_Elements   => Max_Sp_Abs_Time_Elements);

                  --====================================--

    package Relative_Timestamp_Repository is new Generic_Repository
               (Data_Type              => Timestamp_Adt.Relative_Timestamp,
                Default_Value          => Timestamp_Adt.Nil_Relative_Timestamp,
                Max_Data_Elements      => Max_Rel_Time_Elements,
                Special_Pool_Enabled   => True,
                Special_Pool_Address   => Sp_Start_Address,
                Special_Pool_Data_Valid => Sp_Data_Valid,
                Max_Sp_Data_Elements   => Max_Sp_Rel_Time_Elements);

    package Timestamp_Repository renames Absolute_Timestamp_Repository;

                  --====================================--

    procedure Register_As_Consumer
            (The_Item      : in     Configuration_Id_Adt.Repository_Id;
             The_Reference :    out Abstract_Repository.View);


    procedure Register_As_Producer
            (The_Item      : in     Configuration_Id_Adt.Repository_Id;
             The_Reference :    out Abstract_Repository.Reference);


    procedure Unregister
            (The_Item      : in     Configuration_Id_Adt.Repository_Id;
             The_Reference : in out Abstract_Repository.Reference);


    function Number_Of_Elements
            (The_Item : in     Configuration_Id_Adt.Repository_Id) return Integer;


    function Number_Of_Elements
            (The_Item   : in     Configuration_Id_Adt.Repository_Id;
             Start_Time : in     Timestamp_Adt.Relative_Timestamp;
             End_Time   : in     Timestamp_Adt.Relative_Timestamp :=
                                 Timestamp_Adt.Nil_Relative_Timestamp)
             return Integer;

    function Get_Current (The_Item : in Configuration_Id_Adt.Repository_Id) return Data;
```

```
    procedure Get_Buffer
            (The_Item        : in     Configuration_Id_Adt.Repository_Id;
             Number_Of_Items : in out Integer;
             The_Buffer      :    out Item_Buffer);


    procedure Get_Buffer
            (The_Item        : in     Configuration_Id_Adt.Repository_Id;
             Start_Time      : in     Timestamp_Adt.Relative_Timestamp;
             End_Time        : in     Timestamp_Adt.Relative_Timestamp :=
                                      Timestamp_Adt.Nil_Relative_Timestamp;
             The_Buffer      :    out Item_Buffer;
             Number_Of_Items :    out Integer);


    procedure Initialize
            (The_Factory : access Object;
             The_Mode    : in     System_States.Initialization_Mode);



Private --=========================================================================

    type Kind is
            (Floating_Point,
             Signed_Integer,
             Discrete,
             Unsigned_Integer,
             Bounded_String,
             Abs_Timestamp,
             Rel_Timestamp);

    for Kind use (Floating_Point   => 0,
                  Signed_Integer   => 1,
                  Discrete         => 2,
                  Unsigned_Integer => 3,
                  Bounded_String   => 4,
                  Abs_Timestamp    => 5,
                  Rel_Timestamp    => 6);

    for Kind'Size use Interfaces.Unsigned_8'Size;

    type Object is new Fc_Executive_Interface.Object with null record;

end Repository; --=================================================================
```

4.3.15.1 Package Repository Type Declarations

```
type Object    is new Fc_Executive_Interface.Object with private;

    type Reference is access all Object'Class;

    type View      is access constant Object'Class;

    subtype Data_Array is System_Types.Byte_Array (1 .. 4);

    type Data is new Data_Item.Object
        with record
            Value : Data_Array;
        end record;
```

```
    --
    -- The item queue contains the multiple values of a data item.
    --
    type Item_Buffer is array (Integer range <>) of Data;



private

    type Kind is
            (Floating_Point,
             Signed_Integer,
             Discrete,
             Unsigned_Integer,
             Bounded_String,
             Abs_Timestamp,
             Rel_Timestamp);

    for Kind use (Floating_Point   => 0,
                  Signed_Integer   => 1,
                  Discrete         => 2,
                  Unsigned_Integer => 3,
                  Bounded_String   => 4,
                  Abs_Timestamp    => 5,
                  Rel_Timestamp    => 6);

    for Kind'Size use Interfaces.Unsigned_8'Size;

    type Object is new Fc_Executive_Interface.Object with null record;
```

4.3.15.2 Package Repository Package Instantiations

*4.3.15.2.1 **package** Fp_Repository*

The Fp_Repository Class will provide a common data interface for floating point data being passed between the application and the input/output objects. This class is a specialization of the abstract Repository Class extending the concept of the repository for Floating-point data.

```
    package Fp_Repository is new Generic_Repository
                (Data_Type               => Interfaces.Ieee_Float_32,
                 Default_Value           => 0.0,
                 Max_Data_Elements       => Max_Fp_Elements,
                 Special_Pool_Enabled    => True,
                 Special_Pool_Address    => Sp_Start_Address,
                 Special_Pool_Data_Valid => Sp_Data_Valid,
                 Max_Sp_Data_Elements    => Max_Sp_Fp_Elements);
```

*4.3.15.2.2 **package** Int_Repository*

The Int_Repository Class will provide a common data interface for integer data being passed between the application and the input/output objects. This class is a specialization of the abstract Repository Class extending the concept of the repository for integer data.

```
package Int_Repository is new Generic_Repository
            (Data_Type               => Interfaces.Integer_32,
             Default_Value           => 0,
             Max_Data_Elements       => Max_Int_Elements,
             Special_Pool_Enabled    => True,
             Special_Pool_Address    => Sp_Start_Address,
             Special_Pool_Data_Valid => Sp_Data_Valid,
             Max_Sp_Data_Elements    => Max_Sp_Int_Elements);
```

*4.3.15.2.3 **package** Discrete_Repository*

The Discrete_Repository Class will provide a common data interface for discrete data being passed between the application and the input/output objects. This class is a specialization of the abstract Repository Class extending the concept of the repository for discrete data.

```
package Discrete_Repository is new Generic_Repository
            (Data_Type               => Boolean,
             Default_Value           => False,
             Max_Data_Elements       => Max_Dis_Elements,
             Special_Pool_Enabled    => True,
             Special_Pool_Address    => Sp_Start_Address,
             Special_Pool_Data_Valid => Sp_Data_Valid,
             Max_Sp_Data_Elements    => Max_Sp_Dis_Elements);
```

*4.3.15.2.4 **package** Unsigned_Repository*

The Unsigned_Repository Class will provide a common data interface for unsigned data being passed between the application and the input/output objects. This class is a specialization of the abstract Repository Class extending the concept of the repository for unsigned data.

```
package Unsigned_Repository is new Generic_Repository
            (Data_Type               => Interfaces.Unsigned_32,
             Default_Value           => 0,
             Max_Data_Elements       => Max_Unsigned_Elements,
             Special_Pool_Enabled    => True,
             Special_Pool_Address    => Sp_Start_Address,
             Special_Pool_Data_Valid => Sp_Data_Valid,
             Max_Sp_Data_Elements    => Max_Sp_Unsigned_Elements);
```

*4.3.15.2.5 **package** String_Repository*

The Unsigned_Repository Class will provide a common data interface for string data being passed between the application and the input/output objects. This class is a specialization of the abstract Repository Class extending the concept of the repository for string data.

```
package String_Repository is new Generic_Repository
            (Data_Type               => Repository_Types.String_20,
             Default_Value           => (others => ' '),
             Max_Data_Elements       => Max_String_Elements,
             Special_Pool_Enabled    => True,
             Special_Pool_Address    => Sp_Start_Address,
             Special_Pool_Data_Valid => Sp_Data_Valid,
             Max_Sp_Data_Elements    => Max_Sp_String_Elements);
```

4.3.15.2.6 Package Absolute_Timestamp_Repository and Relative_Timestamp_Repository

The Timestamp_Repository Class will provide a common data interface for time data being passed between the application and the input/output objects. This class is a specialization of the abstract Repository Class extending the concept of the repository for timestamp data.

```
package Absolute_Timestamp_Repository is new Generic_Repository
            (Data_Type              => Timestamp_Adt.Absolute_Timestamp,
             Default_Value          => Timestamp_Adt.Default_Absolute_Timestamp,
             Max_Data_Elements      => Max_Abs_Time_Elements,
             Special_Pool_Enabled   => True,
             Special_Pool_Address   => Sp_Start_Address,
             Special_Pool_Data_Valid => Sp_Data_Valid,
             Max_Sp_Data_Elements   => Max_Sp_Abs_Time_Elements);


package Relative_Timestamp_Repository is new Generic_Repository
            (Data_Type              => Timestamp_Adt.Relative_Timestamp,
             Default_Value          => Timestamp_Adt.Nil_Relative_Timestamp,
             Max_Data_Elements      => Max_Rel_Time_Elements,
             Special_Pool_Enabled   => True,
             Special_Pool_Address   => Sp_Start_Address,
             Special_Pool_Data_Valid => Sp_Data_Valid,
             Max_Sp_Data_Elements   => Max_Sp_Rel_Time_Elements);

package Timestamp_Repository renames Absolute_Timestamp_Repository;
```

4.3.15.3 Package Repository Subprograms

*4.3.15.3.1 **procedure** Register_As_Consumer*

This operation will allow a client to register as a consumer of a specified data item. The client is returned a view pointer to the repository holding the data item. This view pointer will only allow the user to read the data item values. The data appears as read only to the client.

```
procedure Register_As_Consumer
        (The_Item      : in     Configuration_Id_Adt.Repository_Id;
         The_Reference :    out Abstract_Repository.View);
```

*4.3.15.3.2 **procedure** Register_As_Producer*

This operation will allow a client to register as a producer of a specified data item. The client is returned a reference pointer to the repository holding the data item. This reference pointer will allow the user to update the data item values. Only one producer is allowed for each data item repository.

```
procedure Register_As_Producer
        (The_Item      : in     Configuration_Id_Adt.Repository_Id;
         The_Reference :    out Abstract_Repository.Reference);
```

*4.3.15.3.3 **procedure** Unregister*

This operation will allow a client to unregister as a producer of an Item. This is required when a client terminates or becomes deallocated.

```
procedure Unregister
        (The_Item      : in     Configuration_Id_Adt.Repository_Id;
         The_Reference : in out Abstract_Repository.Reference);
```

*4.3.15.3.4 **function** Number_Of_Elements {1}*

The Number_Of_Elements operation will return the total number of data values in the repository for the data item.

```
function Number_Of_Elements(The_Item : in Configuration_Id_Adt.Repository_Id) return Integer;
```

*4.3.15.3.5 **function** Number_Of_Elements {2}*

The Number_Of_Elements operation will return the number of data values in the repository for the data item with timestamps within the start and end times specified.

```
function Number_Of_Elements
        (The_Item   : in     Configuration_Id_Adt.Repository_Id;
         Start_Time : in     Timestamp_Adt.Relative_Timestamp;
         End_Time   : in     Timestamp_Adt.Relative_Timestamp :=
                             Timestamp_Adt.Nil_Relative_Timestamp)
         return Integer;
```

*4.3.15.3.6 **function** Get_Current*

The Get operation will return the most current data for the data item. This routine will log an error when called for data types for which the data cannot be converted to four unsigned bytes.

```
function Get_Current (The_Item : in Configuration_Id_Adt.Repository_Id) return Data;
```

*4.3.15.3.7 **procedure** Get_Buffer {1}*

The Get_Buffer operation will return a buffer containing the data for the specified data item. The actual number of items placed in the buffer is returned to the caller along with the buffer. The output buffer is filled starting with the most recent item in the repository. Note Strings are not handled.

```
procedure Get_Buffer
        (The_Item        : in     Configuration_Id_Adt.Repository_Id;
         Number_Of_Items : in out Integer;
         The_Buffer      :    out Item_Buffer);
```

*4.3.15.3.8 **procedure** Get_Buffer {2}*

The Get Buffer operation will return a buffer containing the data for the specified data item for the requested period of time. The specified end time must be no later than the current time. The actual number of items placed in the buffer is returned to the caller along with the buffer. The output buffer is filled starting with the most recent item in the repository. Note Strings are not handled.

If no end time is specified, the current time will be used.

```
    procedure Get_Buffer
            (The_Item        : in     Configuration_Id_Adt.Repository_Id;
             Start_Time      : in     Timestamp_Adt.Relative_Timestamp;
             End_Time        : in     Timestamp_Adt.Relative_Timestamp :=
                                      Timestamp_Adt.Nil_Relative_Timestamp;
             The_Buffer      :    out Item_Buffer;
             Number_Of_Items :    out Integer);
```

*4.3.15.3.9 **procedure** Initialize*

This operation will initialize all software constant repositories and all configuration data specified repositories.

```
    procedure Initialize (The_Factory : access Object;
                          The_Mode    : in     System_States.Initialization_Mode);
```

4.3.16 package Data_Logger

The Data Logger interacts with the rest of the system to provide control of the requests to log data and the subsequent regular recording of this requested data.

```
package Data_Logger is -- ===================================================

    type Object    is tagged null record;
    type Reference is access all      Object'Class;
    type View      is access constant Object'Class;

    Max_Number_Of_Requests : constant Integer := 30;

end Data_Logger; --=========================================================
```

4.3.16.1 Package Data_Logger Type Declarations

```
    type Object    is tagged null record;
    type Reference is access all      Object'Class;
    type View      is access constant Object'Class;
```

For an explanation of the types `Object, Reference`, and View see section 4.3.1.

4.3.17 package Dl_Interface_Types

The Data Logger's interface for providing service routines that can initiate logging requires data types that can be used by other subsystems. This package defines the types necessary to make a log request.

```
with Interfaces;
with Timestamp_Adt;
with Repository;
with Configuration_Id_Adt;
with Data_Validity_Adt;
with System_Types;

package Dl_Interface_Types is -- ================================================

    type Request_Types is (Single, Timed, Free);

    for Request_Types use (Single => 0, Timed  => 1, Free   => 2);

    subtype Log_Update_Rate is Interfaces.Unsigned_32 range 1 .. 1_000; --ms
    subtype Log_Duration    is Interfaces.Unsigned_32 range 0 .. 60_000; --ms

end Dl_Interface_Types; -- ======================================================
```

4.3.17.1 Package DL_Interface_Types Type Declarations

```
    type Request_Types is (Single, Timed, Free);

    for Request_Types use (Single => 0, Timed  => 1, Free   => 2);

    subtype Log_Update_Rate is Interfaces.Unsigned_32 range 1 .. 1_000; --ms
    subtype Log_Duration    is Interfaces.Unsigned_32 range 0 .. 60_000; --ms
```

4.3.18 package Data_Logger.Data_Logger_Interface

The Interface class provides service routines for each subsystem to request the logging of subsystem specific data, a data set, the event itself and/or a text message. If a request was made to freely log a data set, for an indefinite period, the subsystem that requested logging may request termination of the log request.

A subsystem specific service routine is provided to facilitate the use of data formats that are unique to each subsystem.

```
with Dl_Interface_Types;
with Interfaces;
with Configuration_Id_Adt;

use type Interfaces.Unsigned_32;

package Data_Logger.Data_Logger_Interface is -- ================================

    function Log_Data_Set
            (Data_Set     : in     Configuration_Id_Adt.Data_Set_Id;
             Request_Type : in     Dl_Interface_Types.Request_Types   := Dl_Interface_Types.Single;
             Update_Rate  : in     Dl_Interface_Types.Log_Update_Rate := 1_000;
             Time_Before  : in     Dl_Interface_Types.Log_Duration    := 0;
             Time_After   : in     Dl_Interface_Types.Log_Duration    := 0)
            return Natural;

    procedure Stop_Logging_Data_Set (Request_Id : in     Natural);

end Data_Logger.Data_Logger_Interface; --====================================
```

4.3.18.1 Package Data_Logger.Data_Logger_Interface Subprograms

*4.3.18.1.1 **function** Log_Data_Set*

This function is called to initiate the logging of any data set. The Data Set is identified by a configuration id. The request type determines how much data is to be logged for the period (Time_Before to Time_After the current time) set once, at next timeframe the logging process is active. A Request can only be terminated by a specific request to stop logging.

```
    function Log_Data_Set
            (Data_Set     : in     Configuration_Id_Adt.Data_Set_Id;
             Request_Type : in     Dl_Interface_Types.Request_Types   := Dl_Interface_Types.Single;
             Update_Rate  : in     Dl_Interface_Types.Log_Update_Rate := 1_000;
             Time_Before  : in     Dl_Interface_Types.Log_Duration    := 0;
             Time_After   : in     Dl_Interface_Types.Log_Duration    := 0) return Natural;
```

*4.3.18.1.2 **procedure** Stop_Logging_Data_Set*

A specific data set log request can be terminated immediately using the request Id

```
    procedure Stop_Logging_Data_Set (Request_Id : in Natural);
```

4.3.19 Package Vendor_IO

This package defines the routine Log_Private_Data that allows a vendor to log private data packet. The maximum length of a single packet is 4096 bytes as by the constant Max_Data_Size_in_Bytes, the actual size may be less.

The 3PTD will usually write several Private Data packets to the DTU card during a given flight. When the packets are retrieved at the Ground Station, there is no guaranteed way to know the order in which the packets were written to the DTU card. Therefore, it is the responsibility of the caller (i.e. the Vendor) to provide its own internal method of sequencing/packet identification needed to properly reconstruct the data when retrieving the data on the ground station.

```
with Interfaces;

with Timestamp_Adt;
with System_Types;

package Vendor_IO is --=========================================================

    Max_Data_Size_in_Bytes : constant := 4096;

    subtype Cage_Code_Type is System_Types.Byte_Array(1..8);

                  --------------------------------------

    procedure Log_Private_Data
                ( Vendor_Id  : in Interfaces.Unsigned_32;
                  Cage_Code  : in Cage_Code_Type;
                  Timestamp  : in Timestamp_Adt.Relative_Timestamp;
                  Data       : in System_Types.Byte_Array );

end Vendor_IO; --================================================================
```

4.3.19.1 Package Vendor_IO Subprograms

This procedure is used to take an array of bytes and log them to the DTU card. The Vendor_ID is a unique identifier supplied by the 3PTD. The Cage_Code is supplied by the 3PTD and is used post flight to retrieve the data from the ground station. The Timestamp is usually taken at the time of logging or acquisition. It is a relative timestamp, relative to the last DCOL packet written to the card.

If the vendor desires, he can use the routine Absolute_Reference_Timestamp in the package Timestamp_Adt to generate an absolute timestamp and place it directly in the data being written to the card.

```
procedure Log_Private_Data
                ( Vendor_Id  : in Interfaces.Unsigned_32;
                  Cage_Code  : in Cage_Code_Type;
                  Timestamp  : in Timestamp_Adt.Relative_Timestamp;
                  Data       : in System_Types.Byte_Array );
```

4.4 Examples of Using PPU Interfaces Package

The example is comprised of two main classes the Gonkulator (pronounced Gonk-u-lator), and the Imaginary Sensor. Each main class has a factory associated with it. The Gonkulator is a singleton object, it functions as a sensor manager, it simply takes readings from the various sensors, reads a number of repository items pertaining to current flight parameters and then logs private data packets if need be. The factories function as the creators and initializes of the Gonkulator and sensor objects. The ten units of code described are as follows:

The Gonkulator Package Specification
The Gonkulator Package Body

The Imaginary_Sensor Package Specification
The Imaginary_Sensor Package Body

The Gonkulator_Factory Package Specification
The Gonkulator_Factory Package Body

The Sensor_Factory Package Specification
The Sensor_Factory Package Body

The Debug_IO Package Specification
The Debug_IO Package Body

4.4.1 Example Overview

The dynamic behavior of the software always starts with program elaboration; this is an Ada term and describes the initialization of the software system as a whole. Elaboration brings all of the type, object and program unit definitions and declarations into existence, all elaboration completes before the first line of code is executed in the HUMS_Main program unit. This guarantees that all entities exist before they are used anywhere in the software. One aspect of program elaboration that takes place and effects the examples and all HUMS software for that matter is the execution of the sequence of statements (if any) in the package bodies. This is the code between the 'begin', 'end' statements of the package body. The factories make use of this mechanism for their initialization.

Execution of the examples starts with the elaboration of the Gonkulator_Factory and the Sensor_Factory. These factories both perform the same operation; they make a call to Executive.Register to register the factory objects so they can be called during the various phases of initialization.

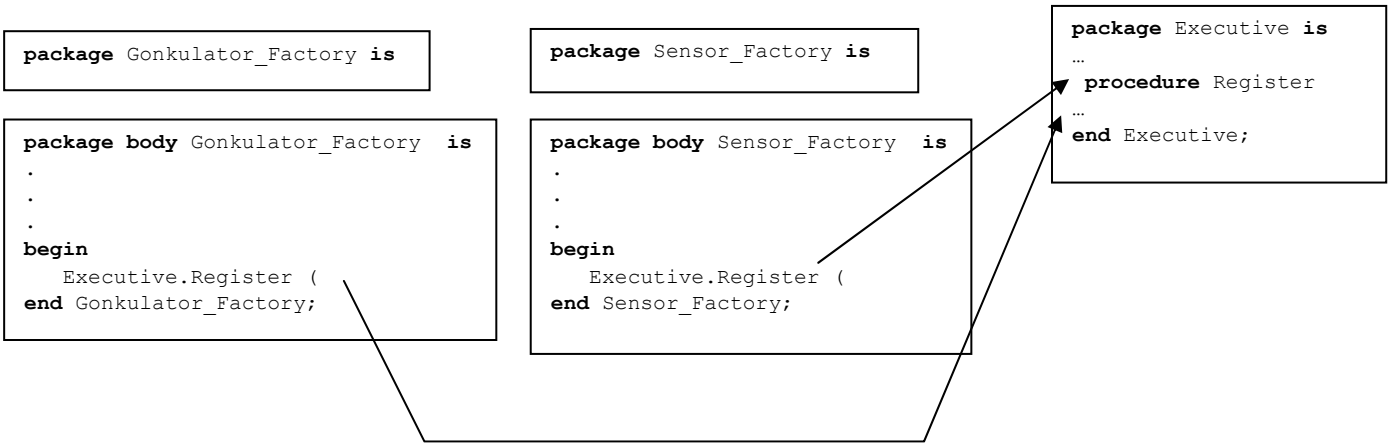


Figure 4-3 Factory Registration with the Executive

The behavior continues with the Executive eventually calling the Initialize routine in the two factories. The Initialize routine in the executive is called three different times, once for each phase of initialization, Creation, Reference Resolution and Hardware Initialization. The factories in turn call the individual routines within the Gonkulator and Imaginary_Sensor packages that correspond to that specific phase of initialization.

It should be noted that the factory bodies contain or hold the actual instances of Gonkulator and all of the instances of the Imaginary Sensors. This is because most uses of the objects only require a reference or pointer to the objects. This is also useful in that the factory is a logical place to dispense reference pointer to the objects when they are needed.

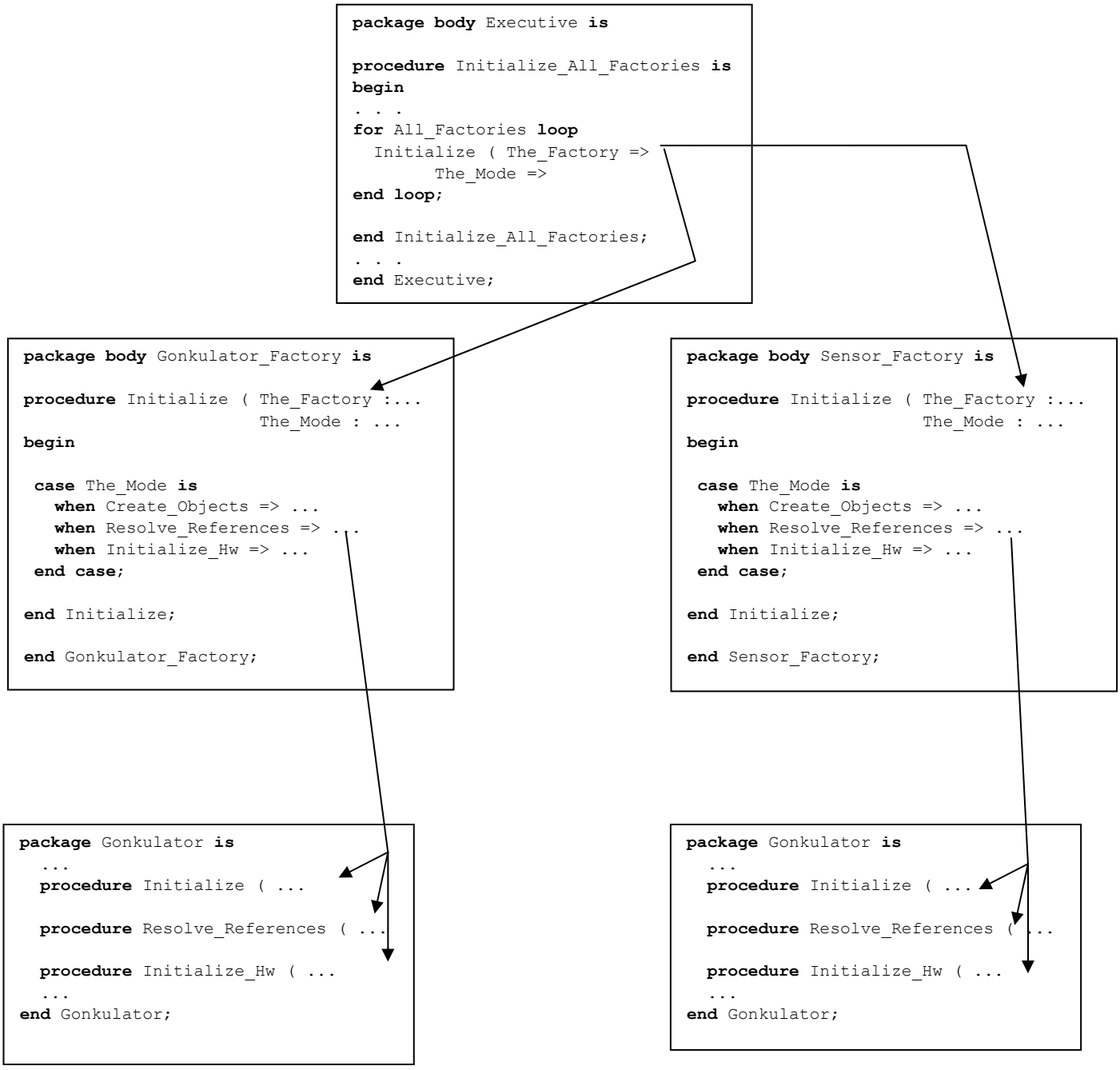


Figure 4-4 Executive Factory Initialization Sequence

The only tricky part of initialization is the phase referred to as Reference Resolution. This phase requires that the Gonkulator gain access to the Sensor objects that it is going to manage. To accomplish this, the `Gonkulator` package body makes a call to the procedure `Sensor_Factory.Sensor_Reference` using a Sensor ID number obtained from its configuration data. This routine returns a pointer (access type) to the Sensor Objects that it will need to collect data from.

The main dynamic behavior of the system is accomplished by calls to the `Update` routines in both the Gonkulator object and the Sensor objects. Within the Update routine the Gonkulator simply reads values from the repository and reads values from the Sensors. The Gonkulator and its Sensor also conduct their reading and writing of the Sensor values via a Semaphore based protocol. The examples make use of the semaphore logic to show the dynamic registering of producers of repository items.

4.4.2 package Gonkulator

```
with System;
with System.Address_To_Access_Conversions;
with Interfaces;

with Cc_Executive_Interface;
with Abstract_Repository;
with Repository;
with Imaginary_Sensor;

package Gonkulator is -- ============================================================

    ---------------------- Type Declarations -----------------------------

    type Object    is new             Cc_Executive_Interface.Object with private;
    type Reference is access all      Object'Class;
    type View      is access constant Object'Class;


    ---------------------- Operation Declarations -------------------------

    procedure Update             (The_Object : access Object);

    procedure Shutdown           (The_Object : access Object);

    procedure Initialize         (The_Object    : access Object;
                                  Base_Address  : in System.Address;
                                  The_Config_Id : in Interfaces.Unsigned_32 );

    function  Is_Initialized     (The_Object    : access Object) return Boolean;

    procedure Resolve_References (The_Object    : access Object);

    procedure Initialize_Hw      (The_Object    : access Object);

private ------------------------------------------------------------------------

   Max_ID_Count : constant := 25;

   type ID_Array is
       array
         (Interfaces.Unsigned_32 range 1..Max_ID_Count)
       of Interfaces.Unsigned_32;

   for ID_Array'Size use Max_ID_Count * 32;

                  ---------------------------------------

   type Ref_Array is
       array
         (Interfaces.Unsigned_32 range 1..Max_ID_Count)
       of CC_Executive_Interface.Reference;
```

```
                --------------------------------------

type Repository_Ref_Array is
    array
      (Interfaces.Unsigned_32 range 1..Max_ID_Count)
    of Repository.Discrete_Repository.Reference;

                --------------------------------------

type Repository_View_Array is
    array
      (Interfaces.Unsigned_32 range 1..Max_ID_Count)
    of Repository.Discrete_Repository.View;

                --------------------------------------

type Value_Array is
    array
      (Interfaces.Unsigned_32 range 1..Max_ID_Count)
    of Interfaces.Unsigned_16;

                --------------------------------------


type Config_Data_Record is
    record
      Vendor_ID               : Interfaces.Unsigned_32;  -- Vendor Id
      Num_of_Sensors          : Interfaces.Unsigned_32;  -- Array Size
      Sensor_ID_List          : ID_Array;                -- Array of sensor config Ids.
      Semaphore_ID_List       : ID_Array;                -- Array of read semaphore Ids.
      Weight_on_Wheels_Id     : Interfaces.Unsigned_32;  -- Repositiory Id
      Indicated_Airspeed_Id   : Interfaces.Unsigned_32;  -- Repositiroy Id
      Fuel_Quantity_Aux_1_Id  : Interfaces.Unsigned_32;  -- Repositiroy Id
      Fuel_Quantity_Aux_2_Id  : Interfaces.Unsigned_32;  -- Repositiroy Id
      Fuel_Quantity_Main_1_Id : Interfaces.Unsigned_32;  -- Repositiroy Id
      Fuel_Quantity_Main_2_Id : Interfaces.Unsigned_32;  -- Repositiroy Id
   end record;

for Config_Data_Record use
    record
      Vendor_ID               at   0 range 0..31;
      Num_of_Sensors          at   4 range 0..31;
      Sensor_ID_List          at   8 range 0..799;
      Semaphore_ID_List       at 108 range 0..799;
      Weight_on_Wheels_Id     at 208 range 0..31;
      Indicated_Airspeed_Id   at 212 range 0..31;
      Fuel_Quantity_Aux_1_Id  at 216 range 0..31;
      Fuel_Quantity_Aux_2_Id  at 220 range 0..31;
      Fuel_Quantity_Main_1_Id at 224 range 0..31;
      Fuel_Quantity_Main_2_Id at 228 range 0..31;
    end record;

for Config_Data_Record'Size use 232 * 8; -- Just to make sure !

-- Allows safe address to pointer conversion for dynamic overlay of data structures.
package Config_Data_Pack is new System.Address_To_Access_Conversions(Config_Data_Record);
```

```
                  --------------------------------------

   type Object is new Cc_Executive_Interface.Object with
     record
       Config_Data              : Config_Data_Pack.Object_Pointer;
       Sensor_Ref               : Ref_Array;
       Semaphore_Ref            : Repository_Ref_Array;    -- Used to register as producer.
       Semaphore_View           : Repository_View_Array;   -- Used to register as consumer.
       Sensor_Value             : Value_Array;
       Weight_on_Wheels_View    : Repository.Discrete_Repository.View;
       Indicated_Airspeed_View  : Repository.FP_Repository.View;
       Fuel_Quantity_Aux_1_View : Repository.Unsigned_Repository.View;
       Fuel_Quantity_Aux_2_View : Repository.Unsigned_Repository.View;
       Fuel_Quantity_Main_1_View: Repository.Unsigned_Repository.View;
       Fuel_Quantity_Main_2_View: Repository.Unsigned_Repository.View;
     end record;

end Gonkulator; --=============================================================
```

4.4.3 package body Gonkulator

```
with System;
with System.Storage_Elements;
with Timestamp_Adt;
with Configuration_Id_Adt;
with Repository;
with Debug_IO;
with Sensor_Factory;
with Vendor_IO;

package body Gonkulator is -- ================================================

  Package_Name : constant String := "Gonkulator";

  Im_Initialized : Boolean := False;

                  ---------------------------------------

procedure Update (The_Object : access Object) is

  Module_Name   : constant String := ".Update";
  The_Semaphore : Repository.Discrete_Repository.Data;

  Weight_on_Wheels     : Boolean;
  Indicated_Airspeed   : Interfaces.IEEE_Float_32;
  Fuel_Quantity_Aux_1  : Interfaces.Unsigned_32;
  Fuel_Quantity_Aux_2  : Interfaces.Unsigned_32;
  Fuel_Quantity_Main_1 : Interfaces.Unsigned_32;
  Fuel_Quantity_Main_2 : Interfaces.Unsigned_32;

  use type Repository.Discrete_Repository.Reference;
  use type Interfaces.IEEE_Float_32;
  use type Interfaces.Unsigned_32;

begin ----------------------------------------

   Debug_IO.Put_Line ( Package_Name & Module_Name & "Start" );

   Sensor_Read_Loop:
   for I in Interfaces.Unsigned_32 range 1..The_Object.Config_Data.Num_of_Sensors loop
       -- Loop throuth the sensors and get their readings !!!
```

```
    if
       Repository.Discrete_Repository.Get_Current
       (The_Object.Semaphore_View(I)).Value = False
    then

      -- Register with the Repository as a Producer of the semaphore.
      Repository.Register_As_Producer
        (The_Item       => The_Object.Config_Data.Semaphore_ID_List(I),
         The_Reference  => Abstract_Repository.Reference(The_Object.Semaphore_Ref(I)));

      if -- Check to see if we registered properly.
         The_Object.Semaphore_Ref(I) /= null
      then

        -- Set the semaphore value.
        The_Semaphore.Value := True;

        -- Write it to the repository.
        Repository.Discrete_Repository.Put
             (At_Repository => The_Object.Semaphore_Ref(I),
              The_Item      => The_Semaphore );

        -- Get the sensors value
        The_Object.Sensor_Value(I) :=
        Imaginary_Sensor.Get_Sensor_Value ( Imaginary_Sensor.Reference(The_Object.Sensor_Ref(I)) );

        -- Now Unregister as a producer of the semaphore.
        Repository.UnRegister
             (The_Item      => The_Object.Config_Data.Semaphore_ID_List(I),
              The_Reference => Abstract_Repository.Reference(The_Object.Semaphore_Ref(I)));

      end if;

    end if;


end loop Sensor_Read_Loop;

-- Read the rest of the repository items.

Weight_on_Wheels     :=  Repository.Discrete_Repository.Get_Current
                           (The_Object.Weight_on_Wheels_View).Value;

Indicated_Airspeed   := Repository.Fp_Repository.Get_Current
                           (The_Object.Indicated_Airspeed_View).Value;

Fuel_Quantity_Aux_1  := Repository.Unsigned_Repository.Get_Current
                           (The_Object.Fuel_Quantity_Aux_1_View).Value;

Fuel_Quantity_Aux_2  := Repository.Unsigned_Repository.Get_Current
                           (The_Object.Fuel_Quantity_Aux_2_View).Value;

Fuel_Quantity_Main_1 := Repository.Unsigned_Repository.Get_Current
                           (The_Object.Fuel_Quantity_Main_1_View).Value;

Fuel_Quantity_Main_2 := Repository.Unsigned_Repository.Get_Current
                           (The_Object.Fuel_Quantity_Main_2_View).Value;
```

```
   -- Process the Reporitory items
   if
     not Weight_on_Wheels
   and then
     Indicated_Airspeed /= 0.0
   and then
     Fuel_Quantity_Aux_1 < 10
   and then
     Fuel_Quantity_Aux_2 < 10
   and then
     Fuel_Quantity_Main_1 < 10
   and then
     Fuel_Quantity_Main_2 < 10
   then

     -- The logging of a private data packet.
     Vendor_IO.Log_Private_Data
        ( Vendor_Id  => The_Object.Config_Data.Vendor_ID,
          Cage_Code  => (8,7,6,5,4,3,2,1),
          Timestamp  => Timestamp_Adt.Relative_Timestamp_Is,
          Data       => (77,97,121,32,68,97,121,32,77,97,121,32,68,97,121) );
                     --  M  a   y     D  a   y     M   a  y      D  a  y
   else
     Debug_IO.Put_Line ( Package_Name & Module_Name & " Life is good ah ? ...." );
   end if;

   Debug_IO.Put_Line ( Package_Name & Module_Name & "End"   );

end Update; ----------------------------------

                  ----------------------------------------
                  --             Shutdown               --
                  ----------------------------------------

procedure Shutdown (The_Object : access Object) is
  Module_Name : constant String := ".Shutdown";
begin ------------------------------------------
   Debug_IO.Put_Line ( Package_Name & Module_Name & "Start" );

   Debug_IO.Put_Line ( Package_Name & Module_Name & "End"   );
end Shutdown; ----------------------------------

                  ----------------------------------------
                  --             Initialize             --
                  ----------------------------------------

procedure Initialize (The_Object    : access Object;
                      Base_Address  : in System.Address;
                      The_Config_Id : in Interfaces.Unsigned_32 ) is

  Module_Name : constant String := ".Initialize";

  use System.Storage_Elements;
  use Configuration_Id_Adt;

  Instance_Id      : Storage_Offset := Storage_Offset(Instance_Is(The_Config_Id));
  Instance_Shift   : Storage_Offset := Boolean'Pos(Instance_Id /= 0);
  The_Data_Address : System.Address := Base_Address;

  begin ----------------------------------------------------------------

   Debug_IO.Put_Line ( Package_Name & Module_Name & " Start" );
```

```
   -- Calculate the address of this objects data item
   The_Data_Address := The_Data_Address +
                       ((Instance_ID - Instance_Shift) *  -- Offset Multiplier
                         Config_Data_Record'Max_Size_in_Storage_Elements);

   -- This takes The_Data address and converts it to a pointer.
   The_Object.Config_Data  := Config_Data_Pack.To_Pointer(The_Data_Address);

   Im_Initialized := True;

   Debug_IO.Put_Line ( Package_Name & Module_Name & " End"   );

end Initialize; ----------------------------------------------------------------

                  --====================================--
                  --           Is_Initialized           --
                  --====================================--

function  Is_Initialized (The_Object : access Object) return Boolean is
  Module_Name : constant String := ".Is_Initialized";
begin --------------------------------------------------------------------
   Debug_IO.Put_Line ( Package_Name & Module_Name & " Start" );

   Debug_IO.Put_Line ( Package_Name & Module_Name & " End"   );
   return Im_Initialized;
end Is_Initialized; ------------------------------------------------

                  --====================================--
                  --          Resolve_References        --
                  --====================================--

procedure Resolve_References (The_Object : access Object) is
  Module_Name : constant String := ".Resolve_References";
begin ------------------------------------------------------

   Debug_IO.Put_Line ( Package_Name & Module_Name & " Start" );

   for I in 1..The_Object.Config_Data.Num_of_Sensors loop

     -- Get all of the references to the Imaginary Sensors.
     Sensor_Factory.Sensor_Reference
         ( Id  => The_Object.Config_Data.Sensor_Id_List(I),
           Ref => The_Object.Sensor_Ref(I) );

     -- Get all of the read only Repository items setup.
     Repository.Register_As_Consumer
         (The_Item       => The_Object.Config_Data.Semaphore_ID_List(I),
          The_Reference  => Abstract_Repository.View(The_Object.Semaphore_View(I)));

   end loop;

   -- Now Register all the rest of the Repositiroy Items.
   -- Register the configuration Id and get back a view.
   Repository.Register_As_Consumer
            (The_Item       => The_Object.Config_Data.Weight_on_Wheels_Id,
             The_Reference  => Abstract_Repository.View(The_Object.Weight_on_Wheels_View));

   Repository.Register_As_Consumer
            (The_Item       => The_Object.Config_Data.Indicated_Airspeed_Id,
             The_Reference  => Abstract_Repository.View(The_Object.Indicated_Airspeed_View));
```

```
   Repository.Register_As_Consumer
            (The_Item       => The_Object.Config_Data.Fuel_Quantity_Aux_1_Id,
             The_Reference  => Abstract_Repository.View(The_Object.Fuel_Quantity_Aux_1_View));

   Repository.Register_As_Consumer
            (The_Item       => The_Object.Config_Data.Fuel_Quantity_Aux_2_Id,
             The_Reference  => Abstract_Repository.View(The_Object.Fuel_Quantity_Aux_2_View));

   Repository.Register_As_Consumer
            (The_Item       => The_Object.Config_Data.Fuel_Quantity_Main_1_Id,
             The_Reference  => Abstract_Repository.View(The_Object.Fuel_Quantity_Main_1_View));

   Repository.Register_As_Consumer
            (The_Item       => The_Object.Config_Data.Fuel_Quantity_Main_2_Id,
             The_Reference  => Abstract_Repository.View(The_Object.Fuel_Quantity_Main_2_View));


   Debug_IO.Put_Line ( Package_Name & Module_Name & " End"   );

end Resolve_References; ------------------------------------

                  ---------------------------------------
                  --           Initialize_Hw           --
                  ---------------------------------------

procedure Initialize_Hw (The_Object : access Object) is
  Module_Name : constant String := ".Initialize_Hw";
begin ------------------------------------------------
   Debug_IO.Put_Line ( Package_Name & Module_Name & " Start" );

   -- If any hardware initializaition is required it takes place here.
   null;

   Debug_IO.Put_Line ( Package_Name & Module_Name & " End"   );
end Initialize_Hw; -----------------------------------


end Gonkulator; --================================================================
```

4.4.4 package Imaginary_Sensor

```
with Interfaces;
with System.Address_To_Access_Conversions;
with Cc_Executive_Interface;

with Repository;

package Imaginary_Sensor is --=================================================

    type Object    is new             Cc_Executive_Interface.Object with private;
    type Reference is access all      Object'Class;
    type View      is access constant Object'Class;

    procedure Update             (The_Object    : access Object);

    procedure Shutdown           (The_Object    : access Object);

    procedure Initialize         (The_Object    : access Object;
                                  Base_Address  : in     System.Address;
                                  The_Config_Id : in     Interfaces.Unsigned_32 );
```

```
    function  Is_Initialized     (The_Object    : access Object) return Boolean;

    procedure Resolve_References (The_Object    : access Object);

    procedure Initialize_Hw      (The_Object    : access Object);


                  --====================================--

   function Get_Sensor_Value ( The_Object : access Object ) return Interfaces.Unsigned_16;



private --=========================================================================

   type Config_Data_Record is
       record
         Sensor_Index  : Interfaces.Unsigned_32;
         Sensor_Name   : String(1..10);
         Semaphore_ID  : Interfaces.Unsigned_32; -- Semaphore Configuration_ID.
       end record;

   for Config_Data_Record use
       record
         Sensor_Index  at  0 range 0..31;
         Sensor_Name   at  4 range 0..79;
         Semaphore_ID  at 14 range 0..31;        -- Semaphore Configuration_ID.
       end record;


   package Config_Data_Pack is new System.Address_To_Access_Conversions(Config_Data_Record);


   type Object is new Cc_Executive_Interface.Object with
     record
       Config_Data    : Config_Data_Pack.Object_Pointer;
       Semaphore_Ref  : Repository.Discrete_Repository.Reference;
       Semaphore_View : Repository.Discrete_Repository.View;
     end record;

end Imaginary_Sensor; --=============================================================
```

4.4.5 package body Imaginary_Sensor

```
with System.Storage_Elements;
with Configuration_Id_Adt;
with Initialization_Manager;
with Repository;
with Abstract_Repository;

with Debug_IO;

package body Imaginary_Sensor is --=========================================


  Package_Name : constant String := "Imaginary_Sensor";

                  --====================================--

  Max_Num_Sensors : constant := 25;
```

```
  -- Class wide data values.
  Sensor_Value : array
                   (Interfaces.Unsigned_32 range 0..Max_Num_Sensors)
                 of Interfaces.Unsigned_16;


                   ---------------------------------------
                   --            Read_Sensor            --
                   ---------------------------------------

procedure Read_Sensor ( Sensor_Index : in  Interfaces.Unsigned_32 ) is
  use type Interfaces.Unsigned_16;
begin ----------------------------------------------------------------

  -- Will be incremented by one each time sensor update is called.
  Sensor_Value(Sensor_Index) := Sensor_Value(Sensor_Index) + 1;

end Read_Sensor; -----------------------------------------------------

                   ---------------------------------------
                   --              Update               --
                   ---------------------------------------

procedure Update (The_Object : access Object) is ---------------------------

  Module_Name   : constant String := ".Update";
  The_Semaphore : Repository.Discrete_Repository.Data;

  use type Repository.Discrete_Repository.Reference;

begin ------------------------------------------------------------------------

   Debug_IO.Put_Line ( Package_Name & Module_Name & "Start" );

   -- Can Only "Read the Sensor" if the semaphore is False.
   if
      Repository.Discrete_Repository.Get_Current
      (The_Object.Semaphore_View).Value = False
   then

     -- Register with the Repository as a Producer of the semaphore.
     Repository.Register_As_Producer
       (The_Item       => The_Object.Config_Data.Semaphore_ID,
        The_Reference  => Abstract_Repository.Reference(The_Object.Semaphore_Ref));

     if -- Check to see if we registered properly.
        The_Object.Semaphore_Ref /= null
     then

       -- Set the semaphore value.
       The_Semaphore.Value := True;

       -- Write it to the repository.
       Repository.Discrete_Repository.Put
            (At_Repository => The_Object.Semaphore_Ref,
             The_Item      => The_Semaphore );

       -- Read this particular Sensor.
       Read_Sensor(Sensor_Index => The_Object.Config_Data.Sensor_Index);

       -- Now Unregister as a producer of the semaphore.
       Repository.UnRegister
            (The_Item      => The_Object.Config_Data.Semaphore_ID,
```

```
            The_Reference => Abstract_Repository.Reference(The_Object.Semaphore_Ref));

     end if;

   end if;

   Debug_IO.Put_Line ( Package_Name & Module_Name & "End"   );

end Update; ------------------------------------------------------------------

                  --====================================--
                  --             Shutdown               --
                  --====================================--

procedure Shutdown (The_Object : access Object) is
  Module_Name : constant String := ".Shutdown";
begin ------------------------------------------
   Debug_IO.Put_Line ( Package_Name & Module_Name & "Start" );

   Debug_IO.Put_Line ( Package_Name & Module_Name & "End"   );
end Shutdown; ----------------------------------

                  --====================================--
                  --             Initialize             --
                  --====================================--

procedure Initialize (The_Object    : access Object;
                      Base_Address  : in     System.Address;
                      The_Config_Id : in Interfaces.Unsigned_32 ) is

  Module_Name   : constant String := ".Initialize";

  use System.Storage_Elements;
  use Configuration_Id_Adt;
  use type Interfaces.Unsigned_32;

  Instance_Id      : Storage_Offset := Storage_Offset(Instance_Is(The_Config_Id));
  Instance_Shift   : Storage_Offset := Boolean'Pos(Instance_Id /= 0);
  The_Data_Address : System.Address := Base_Address;

begin ------------------------------------------------------------------

   Debug_IO.Put_Line ( Package_Name & Module_Name & "Start" );

   -- Calculate the address of this objects data item
   The_Data_Address := The_Data_Address +
                       ((Instance_ID - Instance_Shift) *  -- Offset Multiplier
                         Config_Data_Record'Max_Size_in_Storage_Elements);

   -- This takes The_Data_Address and converts it to a pointer for an overlay.
   The_Object.Config_Data  := Config_Data_Pack.To_Pointer(The_Data_Address);

   Debug_IO.Put_Line ( Package_Name & Module_Name & "End"   );

end Initialize; --------------------------------------------------------
```

```
                  --======================================--
                  --           Is_Initialized           --
                  --======================================--

function  Is_Initialized (The_Object : access Object) return Boolean is
  Module_Name : constant String := ".Is_Initialized";
begin ----------------------------------------------------------------
   Debug_IO.Put_Line ( Package_Name & Module_Name & "Start" );

   Debug_IO.Put_Line ( Package_Name & Module_Name & "End"   );
   return False;
end Is_Initialized; ---------------------------------------------------

                  --======================================--
                  --         Resolve_References         --
                  --======================================--

procedure Resolve_References (The_Object : access Object) is
  Module_Name : constant String := ".Resolve_References";
begin ----------------------------------------------------------------

   Debug_IO.Put_Line ( Package_Name & Module_Name & "Start" );

   -- Register with the Repository as a consumer of the semaphore.
   Repository.Register_As_Consumer
       (The_Item       => The_Object.Config_Data.Semaphore_ID,
        The_Reference  => Abstract_Repository.View(The_Object.Semaphore_View));

   Debug_IO.Put_Line ( Package_Name & Module_Name & "End"   );

end Resolve_References; ----------------------------------

                  --======================================--
                  --          Initialize_Hw             --
                  --======================================--

procedure Initialize_Hw (The_Object : access Object) is
  Module_Name : constant String := ".Initialize_Hw";
begin ----------------------------------------------------
   Debug_IO.Put_Line ( Package_Name & Module_Name & "Start" );

   -- If any hardware initializaition has to take place it can take place here.
   null;

   Debug_IO.Put_Line ( Package_Name & Module_Name & "End"   );
end Initialize_Hw; ---------------------------------


                  --======================================--
                  --          Get_Sensor_Value          --
                  --======================================--

function Get_Sensor_Value
         ( The_Object : access Object ) return Interfaces.Unsigned_16 is
begin ----------------------------------------------------------------
    return Sensor_Value(The_Object.Config_Data.Sensor_Index);
end Get_Sensor_Value; ------------------------------------------------

end Imaginary_Sensor; --=============================================
```

4.4.6 package Gonkulator_Factory

```
with Interfaces;
with FC_Executive_Interface;
with System_States;

package Gonkulator_Factory is --=============================================

    type Object is new FC_Executive_Interface.Object with private;

    procedure Initialize (The_Factory : access Object;
                          The_Mode    : in     System_States.Initialization_Mode);

private --=====================================================================

 type Object is new Fc_Executive_Interface.Object with null record;

end Gonkulator_Factory; --=====================================================
```

4.4.7 package body Gonkulator_Factory

```
with System;
with System_Types;
with Executive;
with Initialization_Manager;
with CC_Executive_Interface;
with Gonkulator;

package body Gonkulator_Factory is --==========================================

    --
    -- The factory is a singleton class, therefore only one object instance is
    -- allowed (defined) for this class.
    --
    This_Factory_Object : aliased Object;

    --
    -- Define the 'reference' variable to be used by this instance when
    -- registering with the executive.
    --
    This_Factory_Reference : Fc_Executive_Interface.Reference := This_Factory_Object'Access;

    Gonkulator_Config_Id : constant Interfaces.Unsigned_32 := 16#0A00_0001#;

    Gonkulator_Object    : aliased Gonkulator.Object;

    Gonkulator_Reference : CC_Executive_Interface.Reference;


procedure Initialize (The_Factory : access Object;
                      The_Mode    : in     System_States.Initialization_Mode) is

 Registration_Status : System_Types.Return_Status;

 The_Data_Address : System.Address;
 The_Data_Lenght  : Natural := 0;

 use type System_States.Initialization_Mode;
```

```
begin ---------------------------------------------------------------------------

   -- Create the Objects that this factory manages !!!
   case The_Mode is  ------------------------------------

     when System_States.Create_Objects =>

        -- Create the reference to the Gonkulator Instance.
        Gonkulator_Reference  := Gonkulator_Object'Access;

       -- Delivers the base address of the Gonkulator Data Table.
       Initialization_Manager.Get_Configuration
             (The_Table        => Gonkulator_Config_Id,
              Pre_Cert_Address => The_Data_Address,
              Pre_Cert_Length  => The_Data_Lenght );

        -- Call the Initialize for the Gonkulator.
        Gonkulator.Initialize (The_Object    => Gonkulator_Object'Access,
                               Base_Address  => The_Data_Address,
                               The_Config_Id => Gonkulator_Config_Id );


        -- Register the Gonkulator with the Executive.
        Executive.Register
            (The_Object     => Gonkulator_Reference,
             With_Config_Id => Gonkulator_Config_Id,
             The_Status     => Registration_Status);

     when System_States.Resolve_References =>

          null;

     when System_States.Initialize_Hw =>

          null;

   end case; ----------------------------------------------



end Initialize; ----------------------------------------------------------------


begin  --=================== Gonkulator_Factory ================================

    --
    -- Register this factory with the executive.
    --
    Executive.Register (The_Factory => This_Factory_Reference);

end Gonkulator_Factory; --=============================================================
```

4.4.8 package Sensor_Factory

```
with Interfaces;
with FC_Executive_Interface;
with CC_Executive_Interface;
with System_States;

package Sensor_Factory is --=====================================================

    type Object    is new FC_Executive_Interface.Object with private;

    procedure Initialize (The_Factory : access Object;
                          The_Mode    : in     System_States.Initialization_Mode);

    procedure Sensor_Reference ( Id  : in  Interfaces.Unsigned_32;
                                 Ref : out CC_Executive_Interface.Reference);


private --=========================================================================

 type Object is new Fc_Executive_Interface.Object with null record;

end Sensor_Factory; --=============================================================
```

4.4.9 package body Sensor_Factory

```
with System;

with System_Types;
with Executive;
with Initialization_Manager;
with Imaginary_Sensor;

package body Sensor_Factory is --==============================================

  --
  -- The factory is a singleton class, therefore only one object instance is
  -- allowed (defined) for this class.
  --
  This_Factory_Object : aliased Object;

  --
  -- Define the 'reference' variable to be used by this instance when
  -- registering with the executive.
  --
  This_Factory_Reference : Fc_Executive_Interface.Reference := This_Factory_Object'Access;

                  --====================================--

  -- There are a total of 12 Sensors defined,(obtained from Goodrich) !!!!
  Sensor_Id_Lo : constant Interfaces.Unsigned_32 := 16#0B00_0001#;
  Sensor_Id_Hi : constant Interfaces.Unsigned_32 := 16#0B00_000C#;

  type Sensor_Info is
    record
      Obj : aliased Imaginary_Sensor.Object;
      Ref : CC_Executive_Interface.Reference;
    end record;

  Sensor_Array : array (Sensor_Id_Lo..Sensor_Id_Hi) of aliased Sensor_Info;

                  --====================================--
```

```
procedure Initialize (The_Factory : access Object;
                      The_Mode    : in     System_States.Initialization_Mode) is

 Registration_Status : System_Types.Return_Status;

 The_Data_Address    : System.Address;
 The_Data_Lenght     : Natural := 0;

 use type System_States.Initialization_Mode;

begin ---------------------------------------------------------------------

   -- Create the Objects that this factory manages !!!
   case The_Mode is  ------------------------------------

     when System_States.Create_Objects =>

       -- Delivers the base address of the Sensor Config Data Table.
       Initialization_Manager.Get_Configuration
             (The_Table        => Sensor_Id_Lo,
              Pre_Cert_Address => The_Data_Address,
              Pre_Cert_Length  => The_Data_Lenght );


        -- There are a number of Sensors, get their references.
        for I in Sensor_Id_Lo .. Sensor_Id_Hi loop

          -- Get the reference for each sensor object.
          Sensor_Array(I).Ref := Sensor_Array(I).Obj'Access;

          -- Call Initialize for each sensor.
          Imaginary_Sensor.Initialize
               ( The_Object    => Sensor_Array(I).Obj'Access,
                 Base_Address  => The_Data_Address,
                 The_Config_Id => I );

          -- Register all of the Imaginary Sensors with the Executive.
          Executive.Register
              (The_Object     => Sensor_Array(I).Ref,
               With_Config_Id => I,
               The_Status     => Registration_Status );

        end loop;

     when System_States.Resolve_References =>  null;

     when System_States.Initialize_Hw => null;

   end case; ------------------------------------------


end Initialize; -----------------------------------------------------------------


procedure Sensor_Reference ( Id  : in  Interfaces.Unsigned_32;
                             Ref : out CC_Executive_Interface.Reference) is
begin ---------------------------------------------------------------------
   Ref := Sensor_Array(Id).Ref;
end Sensor_Reference; -----------------------------------------------------

begin  ---------------------- Sensor_Factory --------------------------------

    --
    -- Register this factory with the executive.
    --
    Executive.Register (The_Factory => This_Factory_Reference);


end Sensor_Factory; -------------------------------------------------------
```

4.4.10 package Debug_IO

```
with Interfaces;

package Debug_IO is  --=====================================================

    function To_Hex ( N : Interfaces.Unsigned_8  ) return String;

    function To_Hex ( N : Interfaces.Unsigned_16 ) return String;

    function To_Hex ( N : Interfaces.Unsigned_32 ) return String;

    procedure Put      ( S : in String );

    procedure Put_Line ( S : in String );

    procedure Draw_Line;

end Debug_IO; --============================================================
```

4.4.11 package body Debug_IO

```
with Interfaces;
with System.Raven_Io;
with Unchecked_Conversion;

package body Debug_IO is --=================================================

 use type System.Bit_Order;

 ------------------------------------------------------------------------
 --              Type needed for Hex Conversion Routines             --
 ------------------------------------------------------------------------
 type Nibbles is mod 16;
 for Nibbles'Size use 4;

                   ---------------------------------------

 type Byte_Array is array (0..1) of Nibbles;
 for Byte_Array'Size use 8;

 type Word_Array is array (0..3) of Nibbles;
 for Word_Array'Size use 16;

 type Double_Word_Array is array (0..7) of Nibbles;
 for Double_Word_Array'Size use 32;

                   ---------------------------------------

  function Conv_US8_to_BA is new
    Unchecked_Conversion ( Interfaces.Unsigned_8, Byte_Array );

 function Conv_US16_to_WA is new
    Unchecked_Conversion ( Interfaces.Unsigned_16, Word_Array );

 function Conv_US32_to_DWA is new
    Unchecked_Conversion ( Interfaces.Unsigned_32, Double_Word_Array );

                   ---------------------------------------

 Map : array (Nibbles range 0..15) of Character := "0123456789ABCDEF";
```

```
 X : Natural := 7 * Boolean'Pos(System.Default_Bit_Order=System.High_Order_First);

         ------------------------------------------------------------
         --               Hex String for Unsigned_32               --
         ------------------------------------------------------------
 function To_Hex ( N : in Interfaces.Unsigned_32 ) return String is
   DWA : Double_Word_Array := Conv_US32_to_DWA(N);
   S   : String(1..13);
 begin ------------------------------------------------------------

 S(1..13) := ('1', '6', '#',
              Map(DWA(abs(X-7))),Map(DWA(abs(X-6))),
              Map(DWA(abs(X-5))),Map(DWA(abs(X-4))),
              '_',
              Map(DWA(abs(X-3))),Map(DWA(abs(X-2))),
              Map(DWA(abs(X-1))),Map(DWA(abs(X-0))),
              '#' );
 return S;

end To_Hex; ------------------------------------------------------


         ------------------------------------------------------------
         --               Hex String for Unsigned_16               --
         ------------------------------------------------------------
 function To_Hex ( N : in Interfaces.Unsigned_16 ) return String is
   WA : Word_Array := Conv_US16_to_WA(N);
   S  : String(1..8);
 begin ------------------------------------------------------------

 S(1..8) := ('1', '6', '#',
             Map(WA(abs(X-3))),Map(WA(abs(X-2))),
             Map(WA(abs(X-1))),Map(WA(abs(X-0))),
             '#' );
 return S;

end To_Hex; ------------------------------------------------------



         ------------------------------------------------------------
         --               Hex String for Unsigned_8                --
         ------------------------------------------------------------
 function To_Hex ( N : in Interfaces.Unsigned_8 ) return String is
   BA : Byte_Array := Conv_US8_to_BA(N);
   S  : String(1..6);
 begin ------------------------------------------------------------

 S(1..6) := ('1', '6', '#', Map(BA(abs(X-1))),Map(BA(abs(X-0))), '#' );

 return S;

end To_Hex; ------------------------------------------------------







         ------------------------------------------------------------
         --                          Put                           --
```

```
        ------------------------------------------------------------
 procedure Put ( S : in String ) is ---------------------------------

 begin --------------------------------------------------------------

   System.Raven_IO.Put (S);

 end Put; -----------------------------------------------------------

        ------------------------------------------------------------
        --                        Put_Line                        --
        ------------------------------------------------------------
 procedure Put_Line ( S : in String ) is ----------------------------

 begin --------------------------------------------------------------

   System.Raven_IO.Put_Line (S);

 end Put_Line; ------------------------------------------------------


        ------------------------------------------------------------
        --                        Put_Line                        --
        ------------------------------------------------------------

procedure Draw_Line is -------------------
  S : constant String := (1..80 => '-');
begin ------------------------------------

   System.Raven_IO.Put_Line (S);

end Draw_Line; --------------------------


end Debug_IO; --================================================================
```

4.5 Availability of PPU SW Templates and Public Class Packages

Third-Party software developers may obtain electronic copies of the PPU Client Class Package, PPU Factory Class Package, and related public packages from Goodrich by contacting the following:

HUMS Contract Manager
Goodrich Aerospace
100 Panton Road
Vergennes, VT 05491
(802) 877-2911

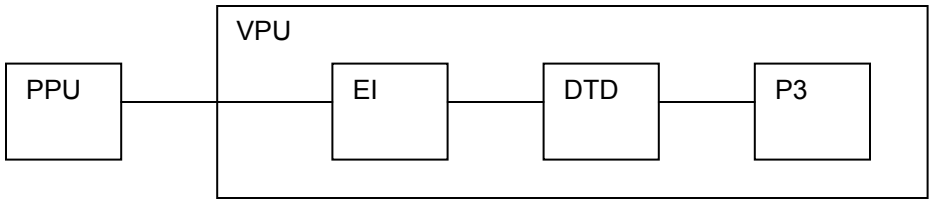
Please reference Goodrich Part Number 3019070 - IMD/COSSI Flight program Ada Specifications.

5 VPU Embedded Software Interfaces

5.1 Overview

The vibration processing unit (VPU) consists of a two-board assembly used to acquire and process vibration (accelerometer) and timing (tachometer) data. One board performs signal acquisition (the data acquisition module - DAM). The sampled data is transferred to a 32 Mbyte DRAM memory area, which is located on the other board (the signal processing module - SPM). Up to eight channels of data may be sampled simultaneously during each acquisition. All signal processing occurs within the signal processing module, executed on two TMS320C31 Digital Signal Processors. The data analysis software is written primarily in the C language, with certain performance-critical functions being written in TMS320C31 assembly language.

There are three major software components within the VPU:



Executive / Interface (EI) - Proprietary software used to control / collect data acquired from the data acquisition module, communicate with the PPU, and act as the main executive for all the VPU software.

Drivetrain Diagnostics (DTD) - Proprietary software used to perform specialized data analyses on the sampled vibration data. Interaction with the EI is through established interfaces.

Third-Party Software (P3) - Software developed by a third-party technology developer (3PTD) used to extend the capabilities of the existing VPU analysis software. It operates as an adjunct to the DTD software. Interaction with the DTD and EI is through established interfaces (detailed in section 5.5.3).

A 3PTD can add functionality to the VPU by using the existing inter-CSCI interfaces to request and obtain raw sensor data and to communicate computational results to the PPU. The following scenario would be typical:

1) The PPU commands an analysis whose identifier corresponds to a P3 function.
2) The VPU EI receives the command and determines that the VPU DTD must supply the relevant acquisition parameters.
3) The VPU DTD determines that the P3 is ultimately responsible, and passes the request along.
4) The VPU P3 supplies the requested acquisition parameters, which are simply passed through the DTD back to the EI.
5) The EI performs the acquisition, and then invokes the DTD to process the raw data.
6) The DTD again simply passes the request along to the P3.
7) The P3 performs the analysis (according to the analysis identifier) and passes its results through the DTD back to the EI.
8) The EI returns the results to the PPU.

5.2 Constraints

The following **hardware** factors constrain the P3 software:

1) Channel multiplexing (section 5.5.4).
2) Magnitude/phase error (section 5.5.2.2, PR_GET_CALIB_FACTOR).
3) Sample rate limitations (section 5.5.4).
4) Memory limitations (section 5.3).
5) Watchdog timer requirements (section 5.4).
6) Multiprocessing synchronization.

The following **software** factors, constrain the P3 software:

1) Inability to process the raw data in real time (section 5.4).
2) Requirement to link (and otherwise peacefully coexist) with the existing EI and DTD CSCIs (section 5.6).
3) Documentation, traceability, and testability requirements (must conform to DO-178B level B).

5.3 Memory

The DSPs in the VPU produce a 24-bit address, and access 32-bit data words. There are three primary memory blocks in the address space, detailed in sections 5.3.1, 5.3.2, and 5.3.3.

5.3.1 Flash EPROM

There is a single 0.5 Mword (512 Kword) block of Flash EPROM, shared by both DSPs. It incurs a nominal 3 wait states for each uncontended access.

It is partitioned into two spaces: 256 Kwords for boot and flight code, and 256 Kwords for configuration data. At power-up/reset, all flight code (including P3) is copied into SRAM, for execution speed.

3PTD's may use up to 64 Kwords for flight code, and 64 Kwords for configuration data.

The P3 configuration data may be accessed as follows:

```
static int *DTD_cfg, *P3_cfg;  /* DTD, P3 config base addresses */
#ifdef MSVC
extern int cfg[];              /* configuration array */
DTD_cfg = cfg + (8 + cfg[4] + cfg[5]);
#else
DTD_cfg = (int *)(0x440000 + *((int *)0x47fff8) + *((int *)47fff9));
#endif
P3_cfg = DTD_cfg + DTD_cfg[12];
```

5.3.2 SRAM

There are two 0.5 Mword (512 Kword) blocks of SRAM, one block dedicated to each DSP. It incurs 1 wait state for each access.

All VPU flight code, stack, and scratchpad data needs to fit within SRAM. 128 Kwords are available for unrestricted P3 usage, including the P3 flight code copied from Flash.

In addition, 3PTD's may use up to 128 Kwords starting at the global symbolic address "ar". However, this memory is not dedicated, in the sense that if another application (EI or DTD) runs an analysis, the memory is overwritten.

However, CONSECUTIVE P3 analyses can depend on this memory remaining intact. An important property of this shared memory area is that it is aligned on a 16 Kword boundary (see section 5.5.2.2, FFT).

The stack is sized at 8 Kwords, of which P3 may consume a maximum of 2 Kwords.

5.3.3 DRAM

There is a single 8.0 Mword (8192 Kword) block of DRAM, shared by both DSPs. It incurs a nominal 3 wait states for each uncontended access.

Of the 8 Mwords, only 7.6 are used for data acquisition. The remaining 0.4 Mword is used for miscellaneous functions, including auxiliary data storage and interprocessor communications. The following memory is available for dedicated P3 usage:

- 42 Kwords in section "INTER_DSP"
- 49 Kwords in section "RES_DSP1"

Just prior to a data acquisition, the EI calculates how much memory is required for each channel based on the requested sample rate and acquisition time, and allocates consecutive, contiguous blocks of DRAM starting from the first available address (the global symbol "GV_SPL"). The only time the raw data section of DRAM is written to is during acquisitions and possibly by the applications. If the same application (P3 for instance) runs CONSECUTIVELY, it can depend on the entire raw data section remaining intact.

5.4 Timing

The VPU does not perform rate-driven processing. Rather, it performs processing-on-demand. As each analysis is commanded from the PPU, an acquisition is performed, data is analyzed, and the results are passed to the PPU.

The DSP's are clocked at 40 MHz, which translates to a 50 nsec instruction cycle (20 MIPS).

All analyses must execute in less than 15 minutes, at which time the watchdog timer resets the processor. Note that BOTH PROCESSORS can be running the entire 15 minutes.

5.5 Interfaces

5.5.1 Definitions

In all subsequent interface definitions, unless specifically stated otherwise, each data item is assumed to be a 32-bit word, whose formatting is explicitly stated. The following formats are used frequently:

- ***Unsigned*** refers to a generic unsigned integer.
- ***Signed*** refers to a two's-complement integer.
- ***Float*** refers to a TMS320C31 single precision floating point value.
- ***2-packed*** refers to a word packed with two 16-bit elements, possibly zero-padded on the right (at the end of a vector of such elements).
- ***4-packed*** refers to a word packed with four 8-bit elements, possibly zero-padded on the right (at the end of a vector of such elements).
- ***Structure*** refers to a generic collection of data requiring further definition.
- ***Vector*** refers to a sequence of identically-formatted data elements.

- ***Xxx+*** refers to a vector of format xxx data elements.
- ***Pointer*** refers to a memory word address.
- ***Xxx**** refers to a pointer to format xxx.
- ***Offset*** refers to an unsigned word offset, relative to the start of the configuration directory (see section 5.3.1). It is used in preference to a pointer, so the configuration table can be relocated easily.

The data structures defined in this section are generic in the sense that they are used to simplify the definition of the specific higher-level data structures in the interface definitions to follow.

An ***analysis_identifier*** (unsigned), is formatted as follows:

- ***type*** (unsigned, bits 31-12): 0x0001C-0x0001F for P3 applications.
- ***index*** (unsigned, bits 11-0): available for P3 use.

An ***identifier*** (unsigned) is formatted as follows:

- ***type*** (unsigned, bits 31-24): available for P3 use.
- ***block count*** (unsigned, bits 23-16): used to allow physically scattered data to be collected as one logical descriptor (see section 5.5.3.2).

index (unsigned, bits 15-0): available for P3 use.

A ***list*** structure is formatted as follows:

- ***count*** (unsigned), of data words to follow.
- ***data*** (padded to word boundary).

A **descriptor** structure is formatted as follows:

- ***identifier*** (as defined above)
- for each block counted in identifier:
  - ***word count*** (signed, > 0 for local data, < 0 for non-local data, = 0 reserved).
  - local ***data***, or ***pointer*** to non-local data.

5.5.2 Software Libraries

5.5.2.1 COTS Libraries

The VPU software may make use of the standard C library functions supported on the TI TMS320C3x processors. These libraries provide standard math and string functions.

5.5.2.2 Other Libraries and Functions

The following functions are defined and implemented in the EI/DTD software, and are available for P3 use:

```
//================================================================
void FFT
  (float    *IOR,   /* (IO) Real input/output vector       */
   float    *IOI,   /* (IO) Imaginary input/output vector  */
   unsigned Log2n,  /* (I ) Log2 (vector size)             */
   int      Dir,    /* (I ) Direction (-1=reverse, 1=forw) */
```

```
   unsigned Mode);  /* (I ) Mode (0=complex, 1=real)       */
//=====================================================================
//  Notes:
//  1)  The IOR and IOI vectors must be located on Log2n-bit address
//      boundaries (to permit hardware bit-reversed addressing).
//  2)  Space for the IOI vector must be supplied even for real
//      transforms (Mode=1), but it need not be initialized.
//=====================================================================

//=====================================================================
void getraw
  (int      *In,    /* (I ) Input vector (2-packed)             */
   unsigned Isize,  /* (I ) # data elements in the input vector */
   float    Mult,   /* (I ) Multiplicative scale factor         */
   float    *Out);  /* ( O) Output vector (n words required)    */
//=====================================================================
//  Notes:
//  1)  Convert 2-packed signed raw input to scaled float output.
//      Mult=1 means output data is scaled from -32768 (-1.25V)
//      to +32766 (+1.25V)).
//  2)  Isize is even (no partial raw data words).
//=====================================================================

//=====================================================================
void PR_GET_CALIB_FACTOR
  (unsigned Sens,     /* (I ) Sensor code (0-61, see section 5.5.4)   */
   unsigned Chan,     /* (I ) ADC channel (0-7, see section 5.5.4)    */
   unsigned Type,     /* (I ) Always set to 4 (ANALOG_CHAIN)          */
   unsigned Gain,     /* (I ) Desired calibration gain setting        */
                      /*      (1, 2, 4, 8,...,128)                    */
   float    Freq,     /* (I ) Desired calibration frequency           */
                      /*      (< 1 kHz)                               */
   float    *Magn,    /* ( O) Correction modulus at input frequency   */
   float    *Phase);  /* ( O) Correction phase at input frequency     */
//=====================================================================
//  Notes:
//  1)  Magn is the multiplier for converting from ADC input (+/- 1.25V)
//      to LRU input voltage (at the specified step-gain/frequency).  It
//      has a nominal value of 1/G, where G is the value in the signal-
//      type / gain table in section 5.5.4.  Note that it does not
//      incorporate the step-gain factor itself.
//  2)  Phase is the additive correction (in radians) at the specified
//      step-gain/frequency.
//=====================================================================

//=====================================================================
int FN_DSP_ID (void)  /* returns the DSP ID (1=master, 0=slave) */
//=====================================================================

//=====================================================================
#define unt unsigned
#define spf float
/* Optimized (assembly) vector routines (i=0; i<N; i++) */
void copy  (void *In, unt N, void *Out);     /* In[i]         --> Out[i] */
void fill  (unt N, spf *Out, spf B);         /* B             --> Out[i] */
void ramp  (unt N, spf *Out, spf B, spf M);  /* M*i+B         --> Out[i] */
```

```
void vadj  (spf *In, unt N, spf B, spf M);   /* M*In[i]+B     --> In[i]  */
void vrev  (void *In, unt N);                /* In[N-i-1]    <--> In[i]  */
void vcmp  (spf *In, unt N, spf *Out);       /* In[i]>=Out[i] --> Out[i] */
void vadd  (spf *In, unt N, spf *Out);       /* Out[i]+In[i]  --> Out[i] */
void vsub  (spf *In, unt N, spf *Out);       /* Out[i]-In[i]  --> Out[i] */
void vmul  (spf *In, unt N, spf *Out);       /* Out[i]*In[i]  --> Out[i] */
void vabs  (spf *In, unt N);                 /* abs(In[i])    --> In[i]  */
spf  vsum  (spf *In, unt N);                 /* sum(In[i])    --> caller */
spf  vmin  (spf *In, unt N);                 /* min(In[i])    --> caller */
spf  vmax  (spf *In, unt N);                 /* max(In[i])    --> caller */
/* Non-optimized (C) vector routines (i=0; i<N; i++) */
void vdiv  (spf *In, unt N, spf *Out);       /* Out[i]/In[i]  --> Out[i] */
void vsqrt (spf *In, unt N);                 /* sqrt(In[i])   --> In[i]  */
void vsin  (spf *In, unt N);                 /* sin(In[i])    --> In[i]  */
void vcos  (spf *In, unt N);                 /* cos(In[i])    --> In[i]  */
//============================================================================
```

5.5.3 CSCI Interfaces

The P3 CSCI is co-resident on the VPU with the DTD and Executive / Interface CSCIs. During normal system operation, the Executive invokes the DTD CSCI, which in turn invokes the P3 CSCI using one of the interfaces defined in this section.

5.5.3.1 P3_Parameters

```
//============================================================================
//PROTOTYPE:
// void P3_Parameters
//   (unsigned  AID,     /* (I ) Analysis identifier                      */
//    unsigned  *Stat,   /* (I ) Status (0=normal, 1=invalid AID)         */
//    unsigned  *IFC,    /* ( O) Indexer frequency counter (note 1)       */
//    unsigned  *AFC,    /* ( O) Accelerometer frequency counter (note 1) */
//    unsigned  *Gain,   /* ( O) 8 actual gain settings (note 2)          */
//    float     *Gadj,   /* ( O) Auto-gain adjustment time (seconds)      */
//    unsigned  *Head,   /* ( O) Auto-gain headroom (#bits), 16 msb's for */
//                       /*      tach, 16 lsb's for accel                 */
//    unsigned  *Offs,   /* ( O) 8 DC offset compensation enables (note 2)*/
//    unsigned  *Sens,   /* ( O) 12 sensor codes (note 3)                 */
//    float     *Acq,    /* ( O) Acquisition time (seconds)               */
//    float     *Proc);  /* ( O) Processing time (seconds)                */
//============================================================================

Notes (see section 5.5.4 for further details):
1)  Hardware counter/divisor settings for establishing the acquisition
    frequency of indexers or accels.
2)  One element for each ADC channel (0-7).  Each Gain element must be set to
    0, 1, 2, 4, 8, 16, 32, 64, or 128;  each Offs element to 0 or 1.
3)  Elements 0-7 correspond to ADC channels 0-7, elements 8-11 must be
    set to 0.
```

5.5.3.2 P3_Analysis

```
//====================================================================
//PROTOTYPE:
//  void P3_Analysis
//    (unsigned AID,    /* (I ) Analysis identifier                */
//     unsigned *Samp,  /* (I ) 12 actual sample counts (note 1)   */
//     unsigned *Gain,  /* (I ) 8 actual gain settings (note 2)    */
//     unsigned *In,    /* (I ) 12 raw data pointers (note 1)      */
//     unsigned *Stat,  /* ( O) 0=normal, 1=invalid AID            */
//     unsigned *Out);  /* ( O) Result descriptor pointers (<1024) */
//====================================================================

Notes (see section 5.5.4 for further details):
1)  Elements 0-7 correspond to ADC channels 0-7, elements 8-11 are not
    used.
2)  One hardware gain setting for each ADC channel (0-7).  It will be
    set to 1, 2, 4, 8, 16, 32, 64, or 128.

The Out parameter points to a 1024-word block which contains a
descriptor count and 1-1023 descriptor pointers.  The aggregate total
block count from all result descriptors must not exceed 4096.

Upon completion of this function, the EI resolves all pointers
encountered in the descriptors, and assembles the referenced data blocks
into a contiguous data stream for transmission to the PPU.  The data
from the slave DSP immediately follows that from the master.

Both instances of P3_Analysis (one per DSP) are invoked by the EI to
support parallel processing.
```

5.5.3.3 P3_Data

```
//====================================================================
//PROTOTYPE:
//  void P3_Data
//    (unsigned AID,    /* (I ) Analysis identifier                */
//     unsigned *Samp,  /* (I ) 12 actual sample counts            */
//     unsigned *Gain,  /* (I ) 8 actual gain settings             */
//     unsigned *In,    /* (I ) 12 raw data pointers               */
//     unsigned PPU1,   /* (I ) Passed directly from PPU command   */
//     unsigned PPU2,   /* (I ) Passed directly from PPU command   */
//     unsigned PPU3,   /* (I ) Passed directly from PPU command   */
//     unsigned PPU4,   /* (I ) Passed directly from PPU command   */
//     unsigned PPU5,   /* (I ) Passed directly from PPU command   */
//     unsigned *Stat,  /* ( O) 0=normal, 1=invalid AID            */
//     unsigned *Out);  /* ( O) Result descriptor (7 words required) */
//====================================================================

Notes:
1)  Elements 0-7 correspond to ADC channels 0-7 (see section 5.5.4),
    elements 8-11 are not used.
2)  One hardware gain setting for each ADC channel (0-7).  See section
    5.5.4 for details.

Parameters PPU1 through PPU5 are passed unaltered from the PPU.
```

Parameters Samp, Gain, and In are supplied by the EI.

5.5.4 Signal Conditioning and Acquisition

The VPU channel multiplexing is arranged as follows (pick one sensor from each column):

ch0	ch1	ch2	ch3	ch4	ch5	ch6	ch7
MRA1	MRA2	MRA3	MRA4	MRA5	MRA6	RT1	RT2
EA1	EA2	EA3	EA4	EA5	EA6	ET1	ET2
DTA1	DTA2	DTA3	DTA4	DTA5	DTA6	DTA7	DTT1
DTA8	DTA9	DTA10	DTA11	DTA12	DTA13	DTA14	DTT2
DTA15	DTA16	DTA17	DTA18	DTA19	DTA20	DTA21	MIC1
DTA22	DTA23	DTA24	DTA25	DTA26	DTA27	DTA28	ET3
DTA29	DTA30	DTA31	DTA32	TRA1	TRA2	OIS1	SIS1

The following sensor abbreviations are used:

RT	rotor tach	MRA	main rotor accel
ET	engine tach	TRA	tail rotor accel
DTT	drive train tach	EA	engine accel
SIS	shaft index sensor	DTA	drive train accel
OIS	optical index sensor	MIC	microphone

The following sensor codes are used in P3_Parameters:

RT	1-2	MRA	13-18
ET	3-5	TRA	19-20
DTT	6-7	EA	21-26
SIS	11	DTA	27-58
OIS	12	MIC	59

The following table gives the signal gain between the LRU connector and the ADC input. The ADC has a full-scale range of [-1.25, 1.25] volts.

Signal Type	Gain
RT,ET,DTT,SIS,OIS	0.0775
MRA,TRA	1.23
EA,DTA	0.25
MIC	0.56

The following table gives the frequency response of each channel type:

Channel Type	High Pass (Hz) -3 dB	Low Pass (Hz) -3 dB
tach/index	-	-
MRAn	0.4	1.7k
TRAn	0.4	3.5k
EAn	0.4	9.5k
DTAn	0.4	56k
MICn	10	56k

```
Note:  for MICn, the -3 dB column is actually -2 dB.
```

Acquisition sample rate in Hz can be set as follows:

$$\frac{20000000}{64*n} \qquad n = 2..64$$

All accels have same sample rate, and tachs can have 1x, 2x, 4x, or 8x the accel sample rate (provided the accel divisor n is divisible by the tach oversample factor (1,2,4,8)).

Gain can be set to "auto", or "fixed", on a channel-by-channel basis. Automatic gain is commanded by setting the "Gain" parameter to 0, and the choices for fixed gain are 1, 2, 4, 8, 16, 32, 64, or 128. The "headroom" for automatic gain control is adjustable as an integer number of bits, using the "Head" parameter. In general, a headroom of N bits means that the gain is set as high as possible, such that the peak signal value does not exceed 2^(-N) of full scale. For each auto-gain acquisition, there is a special adjustment acquisition that precedes it, whose purpose is to sample the current signal strength. The length of the adjustment acquisition is determined by the "Gadj" parameter.

The "Offs" parameter (DC offset compensation enable) is also set independently for each channel. It must be set to 0 for fixed gain, and 1 for automatic gain.

The ADC for each channel stores a 16 bit number into memory (DRAM) for each sample acquired. The msb of this value is not used, and the 15 lsb's are encoded as as two's-complement, representing a voltage range of -1.25V to 1.25V. Two samples are packed into one memory word, the high half being written first.

5.5.5 Sample P3 Application

A sample P3 application which might be embedded into the VPU is presented below. This routine obtains data from the acquistion memory area, performs various conversion and mathematical operations on the data, and returns the result to the DTD CSCI.

The code also details a method for using the established interfaces to transfer limited amounts of data from the PPU to the VPU.

The method entails using the P3_Data interface in a non-standard but perfectly valid way. The PPU end of the transaction would be handled in an action list, and would consist of the following steps:

1) Issue a special "download" acquire and process command, say 0x1FFFF. The P3 code in the VPU would be responsible for intercepting this particular ID, performing a minimal acquisition, and returning a minimal reply. See the example below for particulars.
2) Issue a sequence of data read commands, each containing the "dummy" acquisition ID used in step (1). Each command has three parameters that are not used by the PPU, and these words will contain the P3 initialization data. In the example given below, it is assumed that the low half of the "PPU3" parameter contains the starting offset into a hypothetical initialization array. This offset would be incremented by 3 in each successive data read command.

A maximum of four VPU commands can be issued each second. Since only three words of data are transferred in each command, it takes over eight seconds to transfer 100 words.

To minimize the risk of being aborted due to a capture window change, the initialization sequence should be performed AFTER the main acquisitions, if at all possible. In this case, step (1) can be omitted, and the dummy code in P3_Parameters and P3_Analysis is no longer necessary.

To minimize the time impact of sending this data, all possible VPU processing should be done prior to reception of all the initialization data (during the 250 msec idle period between reception of the next VPU data read command).

```
//=========================================================================
//MODULE DECLARATIONS
//=========================================================================

#include <stdlib.h>
#include <math.h>

#define unt unsigned int
#define spf float

void PR_GET_CALIB_FACTOR (unt Sens, unt Chan, unt Type, unt Gain,
                          spf Freq, spf *Magn, spf *Phase);
void getraw (int *In,  unt Isize, spf Mult,  spf *Out);
void FFT    (spf *IOR, spf *IOI,  unt Log2n, int Dir, unt Mode);

extern spf ar[];      /* 16k-aligned SRAM */
static spf init[99];  /* initialization data from PPU */

//=========================================================================
//FUNCTION DECLARATIONS
//=========================================================================

/*************************************************************/
 void P3_Parameters  /*                                       */
   (unt   AID ,   /* (I ) Analysis identifier                 */
    unt  *Stat,   /* (I ) Status (0=normal, 1=invalid AID)    */
    unt  *IFC ,   /* ( O) Indexer frequency counter           */
    unt  *AFC ,   /* ( O) Accelerometer frequency counter     */
    unt  *Gain,   /* ( O) 8 actual gain settings              */
    spf  *Gadj,   /* ( O) Auto-gain adjustment time (seconds) */
    unt  *Head,   /* ( O) Auto-gain headroom (#bits), 16 msb's */
                  /*      for tach, 16 lsb's for accel        */
    unt  *Offs,   /* ( O) 8 DC offset compensation enables    */
    unt  *Sens,   /* ( O) 12 sensor codes                     */
    spf  *Acq ,   /* ( O) Acquisition time (seconds)          */
    spf  *Proc)   /* ( O) Processing time (seconds)           */
/*************************************************************/
{ unt i;

  *Stat = 0;                /* good status */
  *Gadj = 0.1;              /* gain adjustment time (not used) */
  *Head   = 1;              /* gain headroom (not used) */
  for (i=0; i<8; i++) Gain[i] = 1;     /* fixed unity gain */
  for (i=0; i<8; i++) Offs[i] = 0;     /* no DC offset adjust */
  for (i=0; i<12; i++) Sens[i] = 0;    /* init no channels */

  /* Check for special download acquisition */
  if (AID==0x1ffff)         /* check analysis ID */
  { *IFC = 64;              /* lowest possible... */
    *AFC = 64;              /* ...sample rates */
    Sens[0] = 27;           /* only 1 sensor (DTA1) */
    *Acq = 0.1;             /* short acquisition time */
    *Proc = 0.1;            /* short processing time */
```

```
    return;
  }

  /* Example of typical acquisition setup */
  *IFC = 3;                  /* indexer sample rate (104 kHz) */
  *AFC = 3;                  /* accel sample rate (104 kHz) */
  for (i=0; i<7; i++) Sens[i] = 27+i;  /* sensor code (7 DT accels) */
  Sens[7] = 6;                         /* ...(1 DT tach) */
  *Acq  = 0.1;               /* acquisition time */
  *Proc = 5;                 /* processing time */
}

/*************************************************************/
  void P3_Analysis  /*                                       */
    (unt  AID,      /* (I ) Analysis identifier              */
     unt *Samp,     /* (I ) 12 actual sample counts          */
     unt *Gain,     /* (I ) 8 actual gain settings           */
     unt *In,       /* (I ) 12 raw data pointers             */
     unt *Stat,     /* ( O) 0=normal, 1=invalid AID          */
     unt *Out)      /* ( O) Result descriptor pointers (<1024) */
/*************************************************************/
{ unt i;
  spf mf;             /* multiplicative scale factor */
  spf mc, pc;         /* magnitude and phase corrections */
  spf *r, *xr, *xi;   /* workspace pointers */

  /* Download acquisition or slave DSP, null result */
  if ((AID==0x1ffff) || (FN_DSP_ID() == 0))
  { *Stat = 0;
    Out[0] = 0;
    return;
  }

  /* Set up workspace pointers */
  r  = ar;            /* result block */
  xr = ar + 0x1000;   /* real data */
  xi = ar + 0x2000;   /* imag data */

  /* Compute conversion factor for raw data: */
  /*     2500 mV full-scale ADC range        */
  /*     65536 ADC bits full-scale (getraw)  */
  /*     100 mV/g nominal sensor sensitivity */
  PR_GET_CALIB_FACTOR (27, 0, 4, 1, 100, &mc, &pc);
  mf = (2500 * mc) / (65536 * Gain[0] * 100);

  /* Process scaled raw data */
  getraw ((int *)In[0], 4096, mf, xr);  /* get scaled raw data */
  FFT (xr, xi, 12, 1, 1);               /* compute real forward FFT */
  for (i=0; i<2048; i++) r[i+2] = sqrt (xr[i]*xr[i]+xi[i]*xi[i]);

  /* Format analysis results */
  i = 0x00010000;     /* identifier (1 block)... */
  r[0] = *(spf *)&i;  /* ...store as unt */
  i = 2048;           /* word count... */
  r[1] = *(spf *)&i;  /* ...store as unt */
  *Stat = 0;          /* good status */
```

```
  Out[0] = 1;          /* one pointer... */
  Out[1] = (unt)r;     /* ...to result block */
}

/*************************************************************/
  void P3_Data   /*                                          */
    (unt  AID,   /* (I ) Analysis identifier                 */
     unt *Samp,  /* (I ) 12 actual sample counts             */
     unt *Gain,  /* (I ) 8 actual gain settings              */
     unt *In,    /* (I ) 12 raw data pointers                */
     unt  PPU1,  /* (I ) Passed directly from PPU command    */
     unt  PPU2,  /* (I ) Passed directly from PPU command    */
     unt  PPU3,  /* (I ) Passed directly from PPU command    */
     unt  PPU4,  /* (I ) Passed directly from PPU command    */
     unt  PPU5,  /* (I ) Passed directly from PPU command    */
     unt *Stat,  /* ( O) 0=normal, 1=invalid AID             */
     unt *Out)   /* ( O) Result descriptor (7 words required)  */
/*************************************************************/
{ int i;

  /* Extract initialization data */
  i = PPU3 & 0xffff;          /* mask high half (used by PPU) */
  init[i+0] = *(spf *)&PPU1;  /* transfer...        */
  init[i+1] = *(spf *)&PPU2;  /* ...bit patterns... */
  init[i+2] = *(spf *)&PPU4;  /* ...into floats     */

  /* Format dummy results */
  Out[0] = 0x00010000;      /* identifier (1 block) */
  Out[1] = 1;               /* word count */
  Out[2] = 0;               /* dummy data */
}
```

5.6 Development

The P3 flight code is delivered to Goodrich in the form of one or more object files. These object files must be compatible with those produced by the Texas Instruments C Compiler Toolset, Version 5.0.

The P3 configuration data (if any) is delivered to Goodrich in the form of a binary file. Either endian is acceptable, but must be specified.

Prior to delivery, the 3PTD may choose to use the VPU simulator (supplied by Goodrich) to debug the interfaces and most of the internal software in a PC/Win32 environment.

The simulator environment imposes certain differences on the P3 software:

1) Absolute addresses used in the target (VPU) will not work on a PC (under Win32).
2) TI-specific compiler pragmas, such as "DATA_SECTION", will not be recognized by a typical PC compiler.
3) The C "sizeof" operator returns a word count in the TI compiler, and a byte count in a PC compiler.
4) Configuration data is accessed through pointers at absolute addresses in the target, and through the global symbol "cfg" in the simulator (see section 5.3.1 for details).
5) Raw data is accessed through the global symbol "GV_SPL" in the target, and through the global symbol "raw" in the simulator. Note that indirect access through pointers (like the "In" array in the P3_Analysis interface) works in either environment.

The above differences can conveniently be incorporated in a single source file via conditional compilation.

6 VME Board Interface

The HUMS IMDS OBS can support up to two (2) additional VME boards. These boards would be installed in the Spare A and Spare B slots of the MPU Back plane. These slots accommodate 6U VME boards. Specific interfaces for these boards are defined in the following paragraphs.

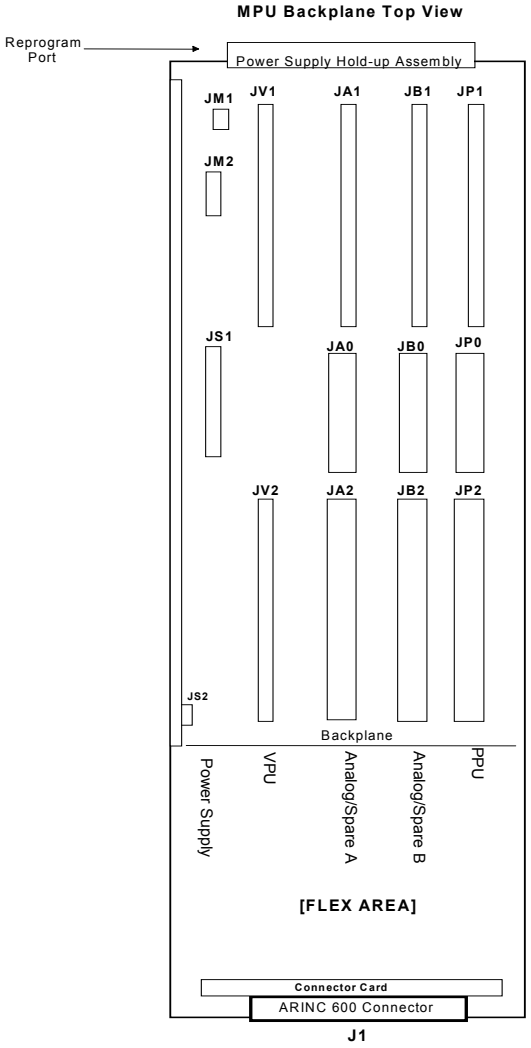


Figure 6-1 Top View of MPU Illustrating the Spare Board Slots

6.1 Power

6.1.1 Voltage

Power to each Spare slot is provided through the VME back plane. The combination of Spare boards may consume no more than 15 W due to thermal considerations. The MPU power supply is designed to provide 5.5 Watts per board on the 5 V supply, and a maximum of 9.5 Watts per board from the 28 V board supply.

6.1.2 Power Up

There are no special power up provisions for boards designed to operate within the Spare board slots.

6.1.3 Power Outage

Each of the Spare board slots is wired to a discrete that signifies eminent power failure. There is a 10 ms power holdup available to allow the boards in the Spare slots to save data, etc. prior to power failure.

6.1.4 Power Dissipation

The combined total worst-case power dissipation for both Spare slots is 15 W. This is primarily a function of the overall system heat dissipation characteristics.

6.2 Mechanical

6.2.1 Spare Boards

Spare boards inserted into the MPU shall be form, fit and function compatible with the VME mechanical standard (see IEEE Standard 1014-1987 and IEEE Standard 1101.2-1992). Spare boards shall be inserted into the MPU chassis from the top as shown in the following figure. Spare boards shall utilize an optional board stiffener, extraction handles and mechanical wedge locks per Goodrich drawing 30190-0458-01. Spare boards shall be of the standard 6U form factor. Spare boards shall be designed for conduction cooling within the MPU chassis in accordance with IEEE Standard 1101.02-1992 and Goodrich drawing 30190-0458-01. See Appendix B (Spare Board Signal for Assignments and Geometry) for detailed mechanical requirements.

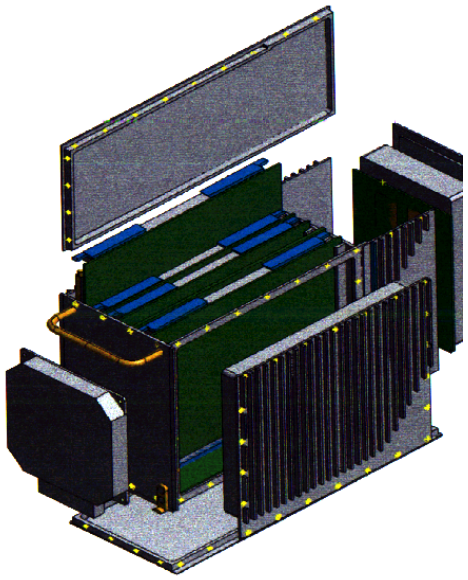


Figure 6-2 Representative MPU Assembly

6.2.2 Deviations

At the present time, there are no mechanical deviations supported within the Spare board design.

6.3 Connectors

6.3.1 Signals

In total, the Spare boards support three connectors:

P1 VME Connector - The standard 3-row, 96 pin VME connector used to support VME data transfer and for board power.

P2 VME Connector - A 5-row, 160-pin VME style connector. The center row is reserved for VME data transfer. The remaining 4 rows are wired primarily to pins on the MPU ARINC-600 connector. The mating back plane connector will support either the recommended 5-row board connector, or the standard 3-row, 96 pin connector. Note that the pin assignments to all 5 rows are predefined. As such, providers of boards in the Spare slots are strongly recommended to use the appropriate 5-row board connector to have access to all the board signals.

P0 VME Connector - A 5-row, 95-pin connector allowed by the VME standard for application-specific I/O. This connector is wired primarily to predefined pins on the MPU ARINC-600 connector. Note that the pin assignments to all 5 rows are predefined. As such, providers of boards in the Spare slots are strongly recommended to use the appropriate 5-row board connector to have access to all the board signals.

6.3.2 Pin Assignments

See Appendix A MPU External Connector Signal Assignment

6.4 Environmental

6.4.1 Temperature/Altitude

The operating environment for the Spare boards is defined from the following:

- DO-160C Category 4(F1), -40 ^{0}C to +55 ^{0}C
- Altitude: Sea Level to 15,000 Feet

The spare boards shall meet the requirements of this specification after exposure to an altitude of 40,000 feet in an un-powered condition.

6.4.2 Temperature Variation

The design of the Spare Boards shall meet the test requirement of DO-160C, Section 5, Category B with low and high operating temperatures from -40 to +55 C.

6.4.3 Shock and Vibration

Spare boards installed into the MPU shall meet the operational requirements of this specification when subjected/exposed to the following test environments:

- DO-160C, Section 8 (Figure 8-4, Curve Y) except the vibration level shall be 5 G's from 52-2000 Hz.
- Operational shock of +/- 15 G's, 3 axes, 1/2 sinusoid of 11 ms duration.
- Crash safety levels of +/- 20 G's in forward/aft, up/down, and lateral (side to side) axes.

6.4.4 Humidity

Spare boards installed into the MPU shall meet Humidity test requirements specified in DO-160C, Section 6, Category B equipment.

6.4.5 Sand and Dust

Spare boards installed into the MPU shall withstand, in both an operating and non-operating condition, exposure to DO-160C Section 12, Category D.

6.4.6 Fungus

Spare boards installed into the MPU shall withstand, in both an operating and non-operating condition, exposure to fungus growth as specified in DO-160C Section 13, Category F.

6.4.7 Salt Atmosphere

Spare boards installed into the MPU shall withstand, in both an operating and non-operating condition, exposure to salt-sea atmosphere as specified in DO160C Section 14, Category S.

6.4.8 EMI

Spare boards installed into the MPU will meet the following EMI requirements:

6.4.8.1 Magnetic Effects

Spare boards installed into the MPU shall be designed to meet the magnetic effect requirement of DO-160C, Section 15 for Class A equipment.

6.4.8.2 Voltage Spike

Spare boards installed into the MPU shall not suffer damage and meet the requirements of this specification when subjected to the test requirements of RTCA/DO-160C Section 17 for Category A equipment.

6.4.8.3 Audio Freq. Conducted Susceptibility

Spare boards installed into the MPU shall not be susceptible to the requirements specified in Section 18 for Category B equipment, except lower range test limit extended to 200 Hz. The test levels shall be expanded from DO-160C levels See Figure 6-3 Audio Frequency Conducted Susceptibility Test Levels.

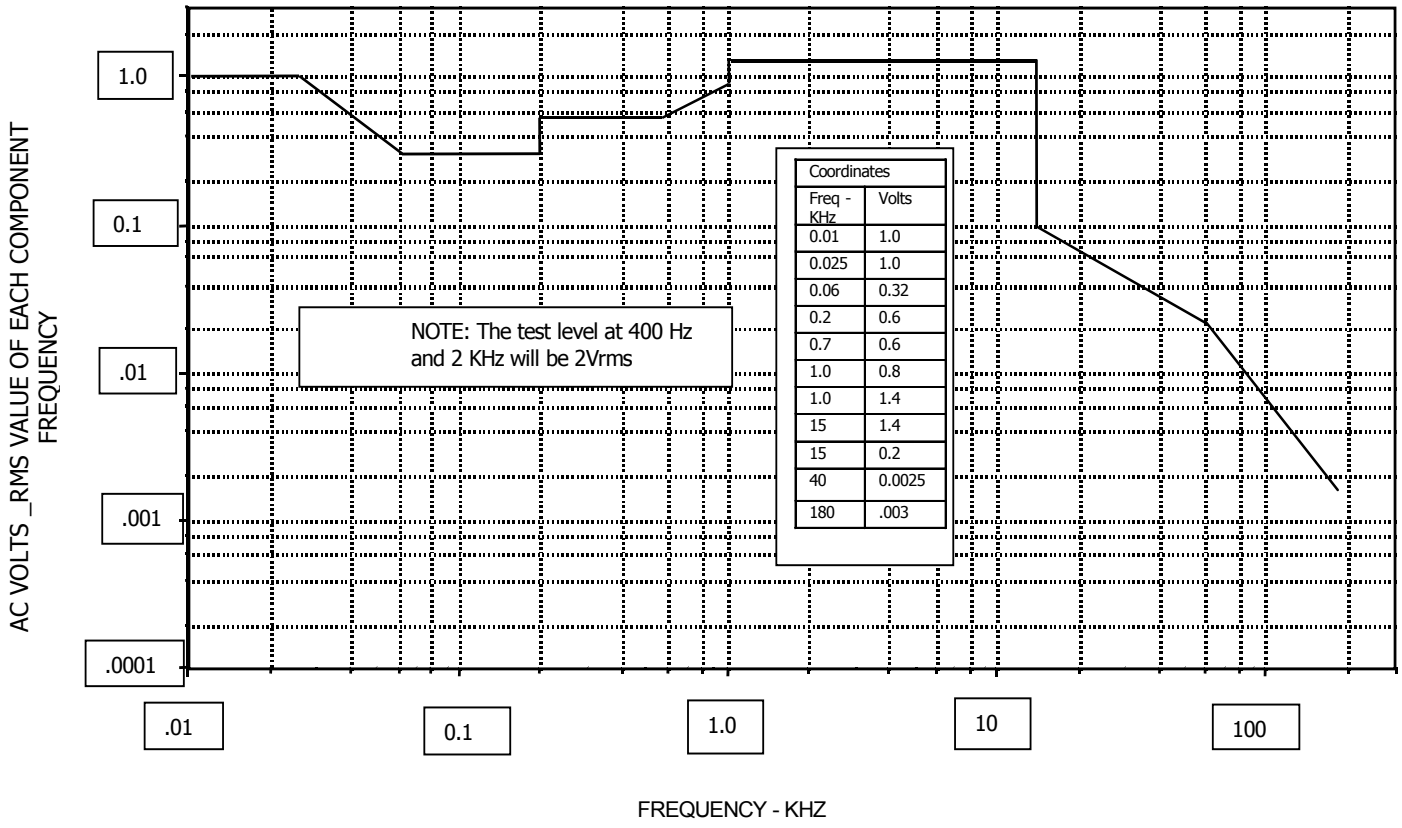


Figure 6-3 Audio Frequency Conducted Susceptibility Test Levels

6.4.8.4 Induced Signal Susceptibility

Spare boards installed into the MPU shall not be susceptible to the EMI test environments defined in RTCA/DO-160C, Section 19 for Category Z equipment with the range extended from 30 Hz - 400 Hz and 15kHZ - 100 kHz at test levels shown in Figure 6-4 RS-101 Magnetic Field Radiated Susceptibility Spec. Limits

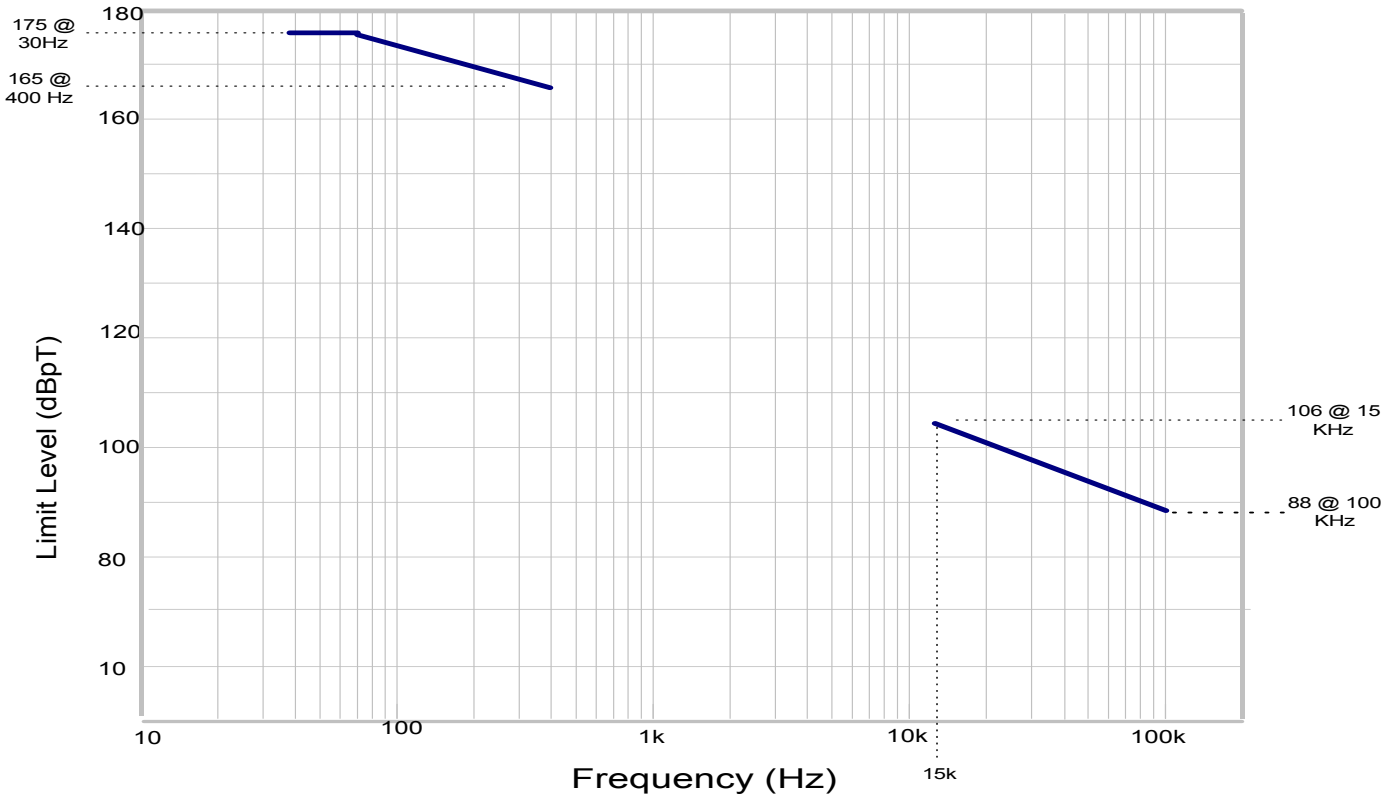


Figure 6-4 RS-101 Magnetic Field Radiated Susceptibility Spec. Limits

6.4.8.5 RF Susceptibility: Radiated and Conducted

Spare boards installed into the MPU shall not be susceptible to the EMI test environments as defined in RTCA/DO-160C, Section 20 for Category Y equipment with scan rates and dwell times in accordance with MIL-STD-461D. Single shield wires will be provided except for +28 VDC power, synchros, and open/ground discretes.

6.4.8.6 Emission of RF Energy

All boards installed into the MPU shall not emit RF noise in excess of the levels specified in RTCA/DO-160C, Section 21 for Category Z equipment except range extended from 10 kHz -150 kHz and 1.215 GHz -18 GHz at levels shown in *Figure 6-5 Radiated Emissions - MIL-STD-461D Lower Frequency Rage* & *Figure 6-6 Radiated Emissions - MIL-STD-461D Higher Frequency Range.* See MIL-STD-461D, profile Figure RE102-2.

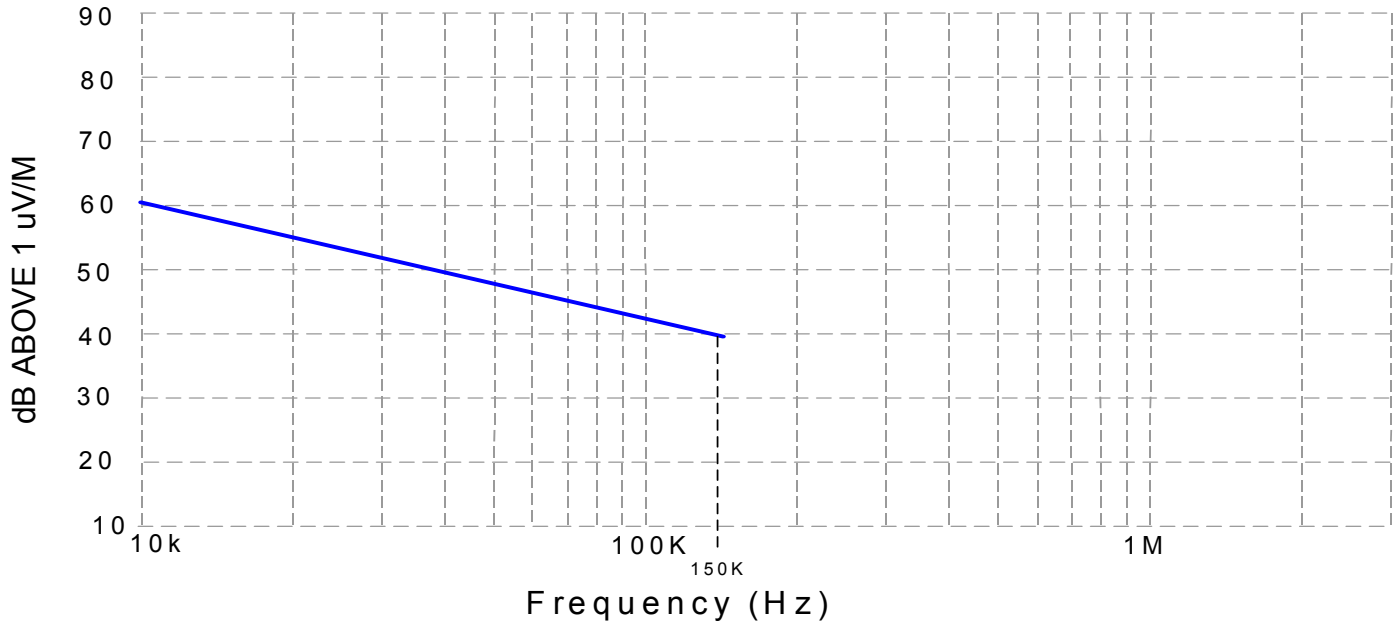


Figure 6-5 Radiated Emissions - MIL-STD-461D Lower Frequency Rage

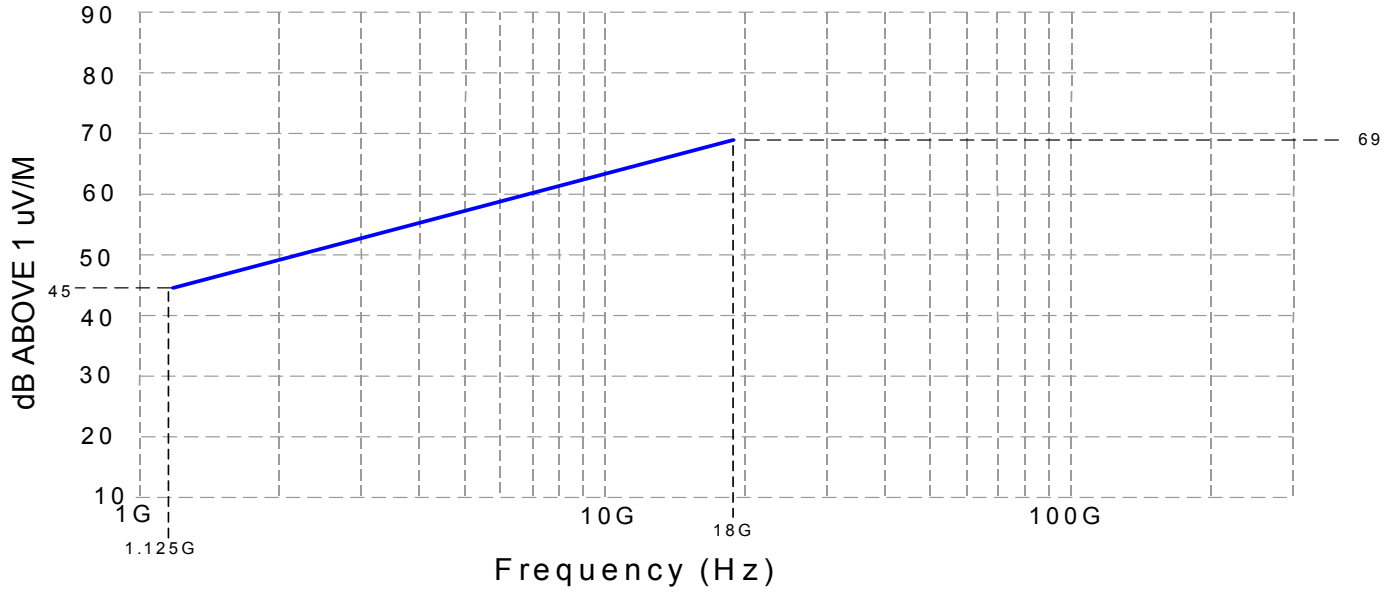


Figure 6-6 Radiated Emissions - MIL-STD-461D Higher Frequency Range

6.4.8.7 Lightning Induced Transients

All boards integrated into the MPU shall operate and not suffer damage following being subjected to the EMI test environments as defined in RTCA/DO-160C, Section 22 for Category XXE4.

The MPU may be upset during the performance of the test but shall recover when the test condition is removed.

6.4.9 Explosion Proof

Boards installed in the Spare slots shall not cause ignition if operated in an explosive atmosphere per DO-160C, Category E, and Environment 2.

6.4.10 Waterproofness

All boards installed into the MPU shall meet the requirements for waterproofness as specified in DO-160C, Section 10 for Category W equipment.

6.5 Bus Interfaces

6.5.1 Bus Types Provided

The Spare Boards of the MPU are each provided with a VME interface.

6.5.2 Access methods, protocols

6.5.2.1 VME Access

The HUMS OBS supports VME bus A24/D32 addressing. VME bus control is provided by the PPU that acts as the VME bus master.

The VME Bus Message interface allows any equipment on the same VME Bus as the PPU system to have read and write access to all the PPU system data.

The VME Bus Messages will be constructed as data areas within the VME Bus address space formatted as specified in the Data Requirements section below. The messages will be created at system initialization and default to an invalid state. The 'data areas' manifest themselves as shared memory between the two memory boards. The slave board must map the area to its physical memory. The bus master will map the area into its own address space. On the masters side this does not necessarily map to physical memory, only into its addressable space.

Concurrent versus Sequential

Subject to the limitations of the VME Bus specification, concurrent access may take place to these message data areas. The messages will all be produced by one CSCI and may be consumed by many CSCI's.

Data integrity is maintained for all fields of the message using the "Update-In-Progress" field. A message producer will mark the message as "Update-In-Progress" before writing, carry out the updating of all fields and then mark the message as "Update-Not-In-Progress".

Communication Protocol

The maximum size of a message, including header and trailing fields, will be 65535 bytes.
Each message will contain a CRC to ensure data integrity.

Data Requirements

The messages that will be transferred and their associated data rates are configurable through the Configuration Data.

Identifier	Description	Units	Range	Accuracy	Precision
Message Identifier	Unique identifier or RITA name	N/A	TBS	N/A	TBS
Data Size	The complete size of the message in bytes.	bytes	TBS	N/A	16 bits
Value	The value of the data formatted appropriately. The format of the value field will be specific to each Message Identifier.	as per Units field	specific to each Message Identifier	N/A	TBS
Units	The units of the Value field	N/A	TBD	N/A	16 bits
Timestamp	The internal P3I software CSCI timestamp This does not represent the time of the VME Bus message.	TBS	TBS	TBS	TBS
Validity	The internal P3I software CSCI validity of the Value field	N/A	TBS	N/A	16 bits
CRC	The CRC calculated TBS	N/A	TBS	N/A	16 bits
Update In Progress	The allows data integrity to be maintained, see note 2	N/A	update in progress update not in progress	N/A	16 bits

6.5.2.2 Signal Assignments

Note that the signal assignments provided in Appendix A are somewhat arbitrary. Most of the signals of the J*0 and J*2 connectors marked as discretes, frequency, excitations, DC low, DC high, etc., are wired to the ARINC 600 connectors on the back of the MPU. As such, they are not presently connected to any specific signals, hence, they can be used to support any additional bus or signal I/O required by a board in the Spare slots.

6.5.3 Limitations: Timing, Bandwidth, etc.

The limitation, such as timing, bandwidth, will be identified as part of the process of integrating the third party technology into the MPU. Completion of the Technology Integration Questionnaire is the starting point to identification of limitations associated with the system and the third party's technology

6.6 Configuration Data

Communication between the Spare board slots and the PPU (and hence the Data Repository, Datalog function, etc.) is configured through the Configuration Data Tables. Providers of VME boards, which are to operate within the MPU, need to carefully coordinate their needs with the HUMS systems integrator.

7 MPU External Interfaces

7.1 Connectors

All I/O to the MPU is made through an ARINC 600 connector. The back view of the MPU connector is shown in the following figure.

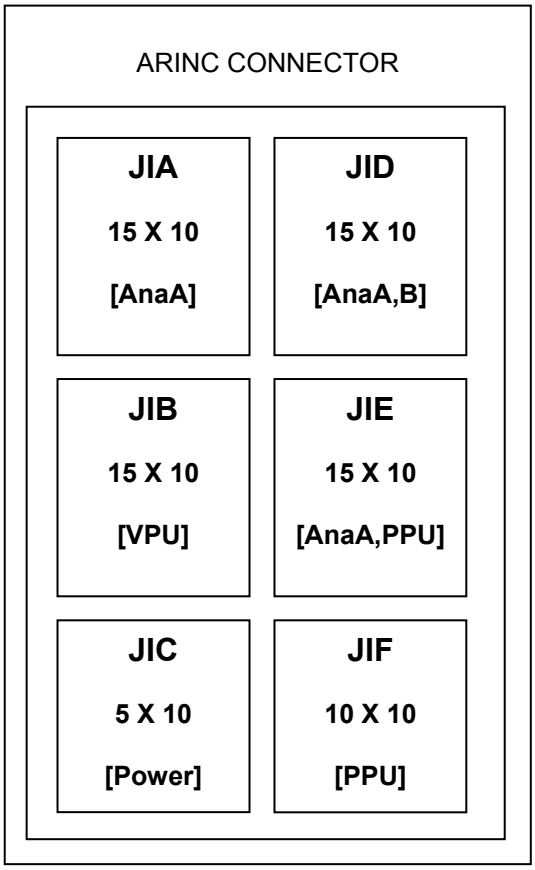


Figure 7-1 Back View of MPU ARINC 600 Connector

7.1.1 Signals

The MPU supports interfaces to the following signal types:

Discrete Input/ Output

Analog Input, including

- Analog Differential
- Analog Single Ended

Synchro

Accelerometer

Frequency

Tachometer

Power

Optical Tracker Pulse

Index Sensor

RTD Sensor

Each of these signal types are brought into the MPU through predefined pins. For those signals which are spare, the signal type is arbitrary. Refer to the following section to see the pin assignments.

7.1.2 Signal Pin Assignments

See Appendix A.

7.2 Bus Interfaces

7.2.1 Bus Types

The MPU provides external interfaces to four (4) separate data busses:

- ARINC 429 - The MPU supports a total of 14 ARINC 429 receivers and 4 ARINC 429 transmitters. Either low-speed or high-speed ARINC 429 communication is supported.
- MIL-STD-1553 - The MPU supports a single bus MIL-STD-1553, configured to either the 1553A or 1553B variant. The MPU can be configured to be either an RT or a bus monitor. The specific configuration is limited by aircraft specific availability.
- ARINC 717 - The MPU provides a single ARINC 717 transmitter for interfacing with instruments such as flight data recorders. In addition, there is a single ARINC 717 repeater channel available.
- RS-422 - The MPU supports 7 RS-422 channels. Subsets of the channels can be configured as RS-232 or RS-485 busses.

These interfaces are not directly accessible to third party developers.

The amount of available (unused) serial data channels is limited by aircraft specific needs.

7.2.2 Access Methods, Protocols

Communications via the serial data busses is covered by the appropriate bus standards.

7.2.3 External Bus Interface Pin Assignments

See Appendix B, Spare Board Signal Assignment & Geometry

7.2.4 Limitations: Timing, Bandwidth, etc.

The limitation, such as timing, bandwidth, will be identified as part of the process of integrating third party technology into the MPU. Completion of the Technology Integration Questionnaire is the starting point to identification of limitations associated with the system and the third party's technology..

7.3 Configuration Data

I/O to the MPU is configured through the MPU Configuration Data Tables. This includes both signal data and bus data. The development of these tables require that the technology provider and the HUMS systems integrator to work together to obtain a proper definition of the interface.

8 Ground Station Interfaces

For the interfaces through which communication with the Ground Station application is achieved, please see the following documents.

- 6000051-01-ICD-0101, Interface Control Document for the Health and Usage Management System Activity Data File Component
- 6000051-45-ICD-0101, Interface Control Document for the HUMS Task Controller Component

9 HUMS Systems Integration Process Model

9.1 Problem Domain

The HUMS is defined to be a generic data collection and processing systems intended primarily for use with aircraft health and usage monitoring. It has been designed to be fully configurable. As such, each aircraft utilizing a HUMS will have its own unique configuration data. This data ultimately is used to configure / define the contents of the PPU Data Repository, MPU processing, DTU data logging, and Ground Based System data processing and database content. Within the domain of aircraft health and usage monitoring, it should be recognized that the HUMS will typically deal with certain types of information:

For Example: Exceedance and Usage Data

- Airspeed
- Pitch Attitude
- Roll Attitude
- Heading Attitude
- Vertical Acceleration
- Lateral Acceleration
- Longitudinal Acceleration
- Main Rotor Speed
- Engine Torque(s)
- Gas Generator Speed(s)
- Turbine Gas Temperature(s)
- Pressure Altitude
- Radar Altitude
- Cyclic Longitudinal
- Cyclic Lateral
- Collective
- Pedal
- Pitch Rate
- Roll Rate
- Yaw Rate
- True Airspeed
- GPS Lat/Long Position
- GPS Lat/Long Velocity
- Inertial Nav Lat/Long Position
- Inertial Nav Lat/Long Velocity
- OAT
- Fuel Qty's
- Total Fuel Weight
- Cargo Weight
- Single Point Sling Weight
- Hook Release
- Sling Hoist
- WOW
- MR Brake
- Rotor Fold
- Stores Configuration
- Bat Bus Voltage

- Gen Amps
- Engine Ice On
- Transmission Oil Cool Discretes
- Inverter Overtemp
- Hydraulic Temp/Press
- Generator Fail
- Ice Detect Discrete
- APU Fail
- APU ON
- APU High Oil Temp
- Battery Temp

Engine Performance Data

- Fan Speed(s)
- Gas Generator Speed(s)
- Turbine Gas Temperature(s)
- Engine Torque(s)
- Oil Pressure(s)
- Oil Temperature(s)
- Bleed Valve State(s)
- Eng Chip(s)
- Oil Bypass Discrete(s)
- Fuel Bypass Discrete(s)

Rotor Track & Balance Data

Typically, raw accelerometer data in not normally directly available in the Data Repository. The HUMS VPU preprocesses this data into a format used by the HUMS GBS for Rotor Track and Balance (RTB) and related vibration data analysis. This data can be made available in the Data Repository for download, but this action is limited by system timing and bus speed factors.

Drive Train Data

Drive train accelerometer data is not normally directly available in the Data Repository. Like the Rotor Track and Balance Data, it is preprocessed by the VPU. This data can be made available in the Data Repository for download, but this action is limited by system timing and bus speed factors. In addition, a number of drive train related parameters may be found in the Data Repository:

- Combining Gearbox Oil Press
- Combining Gearbox Oil Temp
- Combining Gearbox Low Oil Temp
- Combining Gearbox Chip
- Tail Gearbox Oil Press
- Tail Gearbox Oil Press
- Tail Gearbox Chip
- Intermediate Gearbox Oil Press
- Intermediate Gearbox Oil Temp
- Intermediate Gearbox Chip
- Main Gearbox Oil Press
- Main Gearbox Oil Temp
- Main Gearbox Low Oil Temp
- Main Gearbox Chip

- Main Gearbox Low Oil Level

For each publicly available parameter in the Data Repository, the following information will be made available:

A/C Type	Signal	Data Repository Name	ID	Eng. Units	Min	Max	Accur.	ate (Hz)

Though each aircraft utilizing a HUMS will work with this type of data, there is no generic / standard list of parameters within the Data Repository. Each vehicle will have its own unique flavor (source, update rate, accuracy / resolution, etc.) of information. As such, a Third-Party Technology Provider needs to become acquainted with the specific HUMS information configured for his respective aircraft. Appendix C details current / future aircraft slated to utilize the Goodrich HUMS. Technology Providers may obtain a list of available aircraft-specific parameters within the HUMS Data Repository by contacting:

HUMS Contract Manager
Goodrich Aerospace
100 Panton Road
Vergennes, VT 05491
(802) 877-2911

9.2 Responsibilities

Successful insertion of new technology in the HUMS requires close coordination between the HUMS Systems Integrator and the Third-Party Technology Provider(s). Simply stated, it is the responsibility of the HUMS Systems Integrator to insure that third-party technology can be efficiently inserted without reducing the operational objectives of the HUMS. It is the responsibility of the Third-Party Technology Provider to define his interface / data needs required to interface new technology within the HUMS. The following table lists the responsibilities of the HUMS Systems Integrator and the Technology Providers:

PHASE	HUMS SYSTEMS INTEGRATOR	TECHNOLOGY PROVIDER
ANALYSIS	Review / Approve Power Needs	Define Power Needs
	Review / Approve Memory Needs	Define Memory Needs
	Review / Approve Mech. Interfaces	Define Mechanical Interfaces
	Review / Approve Timing Needs	Provide Initial Timing Needs
	Provide Data Repository ID, etc.	Request Data Availability
	Review / Approve Spec.	Provide Interface Specification(s)
	Review System Safety	Support Safety Review
DESIGN	Provide Design Support	Product Design
	Update Configuration Data	Provide Config. Data Needs
INTEGRATION	-	Provide Hardware / Software Modules
	Build / Compile / Integrate System	Support Build / Compilation
	Perform Bench Testing	Support Bench Test
	Configure Test System	-
	Perform Regression Testing	-
SYSTEM TEST	Review Test Descriptions	Provide Test Descriptions
	Support Test Proc. Development	Test Procedure Development
	Perform System Testing	Support System Testing
INSTALLATION	Install New Technology	Support Product Installation
FLIGHT TEST	-	Provide Test Procedures
	Support Flight Test	Monitor Flight Tests
	-	Review Test Results
POST FLIGHT	Reconfigure System	Remove Hardware

9.3 Typical Systems Integration Scenarios

Examples of anticipated technology insertion activities are presented below. These examples illustrate the process by which the Third-Party Technology Developer (3PTD) and the HUMS Systems Integrator (HUMS SI) coordinate the insertion of new technology into the HUMS. Specific issues to be discussed between the 3PTDs and the HUMS SI are outlined in Appendix D. As both the IMD HUMS and the products of the 3PTDs contain proprietary information. Though the HUMS system (as demonstrated by this specification) provides a very crisp boundaries between core HUMS functions and functions of 3PTDs, appropriate controls must be exercised amongst all parties during technology insertion.

9.3.1 External Third-Party Box Communicating with MPU via ARINC 429

Scenario: A Technology Provider desires to have a separate box communicate with the HUMS over unused ARINC 429 transmitters / receivers. The box received various data from the HUMS and performs processing on the data. The box transmits data back to the HUMS for display on a CDU and for logging to the DTU.

The HUMS SI is not directly involved in the electrical or mechanical installation / interface with of the new box. Instead, the HUMS SI provides information to the 3PTD as to the availability of desired information from the Data Repository. The 3PTD defines ARINC message content (labels, data types, bus speed, etc.) in an interface spec to the HUMS SI. The HUMS SI reviews / approves the spec and uses this as a basis for modifying the configuration database. The 3PTD details which of the transmitted labels need to be logged / displayed, logging rates, CDU display definitions, and data repository needs. Again, the HUMS SI configures these elements in the configuration database. Once the system is configured, the HUMS SI loads the configuration data into the MPU, and sets up the test bench. The 3PTD provides the new box that is integrated with the MPU. After successful bench testing, the box is installed on the test aircraft and flight test is performed. After test it is not necessary for the HUMS SI to have any detailed knowledge about the technology within the 3PTD's box.

9.3.2 Third-Party Application in PPU

Scenario: A 3PTD wants to embed an application within the PPU to perform some function. It will make use of information within the Data Repository, and in turn, place new data into the Data Repository for logging to the DTU. The HUMS SI provides information to the 3PTD as to the availability of the required data. The 3PTD provides a specification / description of the new data to be placed into the repository. The HUMS SI updates the Configuration Database with the new information. The 3PTD develops their Client and Factory Class Packages. The HUMS SI integrates / compiles the new classes into the HUMS Flight Program CSCI. At this point, the HUMS SI is responsible for maintain appropriate configuration management of the developed product. The HUMS SI coordinates with the 3PTD to insure proper scheduling of the 3PTD tasks is accomplished. The HUMS SI performs initial timing / memory analyses of the new Flight Program. The 3PTD also details data logging needs that the HUMS SI configures in the Configuration Database. After successful testing within the Window NT development environment, the Flight program is cross-compiled to the PowerPC target and loaded into the MPU with the updated Configuration Data. At this point integration and testing proceed, ultimately leading to flight-testing.

If the 3PTD needs to completely hide the source code from the HUMS SI, they may use an identical development system to generate both the Windows NT and PowerPC object files. These would then be linked into the Flight Program as necessary.

9.3.3 Third-Party VME Board in MPU

Scenario: A 3PTD desires to insert a VME board into one of the spare slots within the MPU. This board would operate semi-autonomously and provide I/O to/from the Data Repository in the PPU via the VME bus. The board would acquire I/O externally via spare signals pins on the MPU ARINC 600 connector.

The 3PTD would define its data needs (input and output) associated with the Data Repository. The HUMS SI would configure the Configuration Database to add the additional data required by the 3PTD as well as extend the Data Repository to map into the VME address space. The 3PTD will need to provide thermal and power requirements to the HUMS SI to insure that the VME board will operate within the thermal and power constraints of the MPU. The HUMS SI will coordinate with the 3PTD to insure that cabling to the ARINC connector is properly defined. As necessary, the HUMS SI will modify the test equipment & cable harnesses to support the new I/O to the VME board. The 3PTD will provide the board to the HUMS SI for engineering integration and test. The HUMS SI will work with the 3PTD to assess system safety issues associated with inserting the VME board into the MPU. The HUMS SI will document hardware and software configuration management for the configured MPU. After installation of the requisite aircraft modifications and the reconfigured MPU, the 3PTD and the HUMS SI will support flight test.

9.3.4 Third-Party Application Accessing Ground Station ADF(s)

Scenario: A 3PTD desires to access information collected on a DTU. The 3PTD will then use this information within a Windows NT-based application that performs some post-flight data processing.

The 3PTD would initially request simulated data be logged to a DTU. The HUMS SI would work with the 3PTD to generate representative data on a DTU. This data would then be converted to an ADF for use by the 3PTD for software development. The HUMS SI would also provide the ADF DLLs for use by the 3PTD. Once actual flight data is collected on the DTU, it is stripped from the DTU and ultimately made available from the HUMS GBS. Either the ADF can be served / transferred to a non-GBS computer for processing, or accessed from the GBS computer. The 3PTD can then access the ADF(s) and use them in its processing. If the 3PTD desires to utilize the HUMS GBS, then configuration management of the GBS software (including the 3PTD software) follows under the control of the HUMS SI.

10 Notes

10.1 Abbreviations & Acronyms

A/C	Aircraft
ADF	Activity Data File
BIT	Built in Test
CDU	Cockpit Display Unit
CRC	Cyclic Redundancy Check - a data integrity method that provides a high degree of integrity for data transmission
COM	Common Object Model - Microsoft standard for providing API services to executables and objects.
CSC	Computer Software Component - A portion of a CSCI that performs a major function or a group of related functions. A CSC is composed of one or more Units.
CSCI	Computer Software Configuration Item - A separately releasable and controlled software element (many times what would be considered a program). A CSCI is composed of one or more CSCs
CSU	Computer Software Unit - Also called a Unit, one or more CSUs make up a CSC
DLL	Dynamic Link Library
DTU	Data Transfer Unit - A PCMCIA card device used to read and write data for the transfer to or from the HUMS onboard system to the HUMS Ground Station
ECO	Engineering Change Order
EMI	Electro Magnetic Interference
GBS	Ground Based Software - the main Ground station application
GS	Ground Station
HUMS	Health and Usage Management System - The overall system consisting of the onboard system, ground based system and associated interfaced systems.
HUMS SI	HUMS Systems Integrator
IMDS	Integrated Mechanical Diagnostics System
IRS	Interface Requirements Specification
MPU	Main Processor Unit - The onboard HUMS component that collects data, provides on board analysis, controls onboard display functions and send data to the DTU for transfer to the Ground Station
OBS	Onboard System
ODBC	Open Database Connectivity
PGS	Portable Ground System
PPU	Primary Procession Unit - The core processor in the MPU
PWB	Printed Wiring Board
RDF	Raw Data File
RITA	Rotorcraft Industry Technology Association - A US competitiveness group whose principal members are Boeing Philadelphia, Boeing Mesa, Sikorsky Aircraft, and Bell Helicopter
RTB	Rotor Track and Balance also Rotor Trim and Balance
RTM	Requirements Traceability Manager - A software tool
S/W	Software
StP	Software Through Pictures
TBS	To Be Supplied
VME	A Backplane Bus Standard.
VPU	Vibration Processing Unit - A coprocessor in the MPU that perform vibration acquisition and data reduction.
3PTD	Third-Party Technology Developer

Appendix A MPU External Connector Signal Assignment

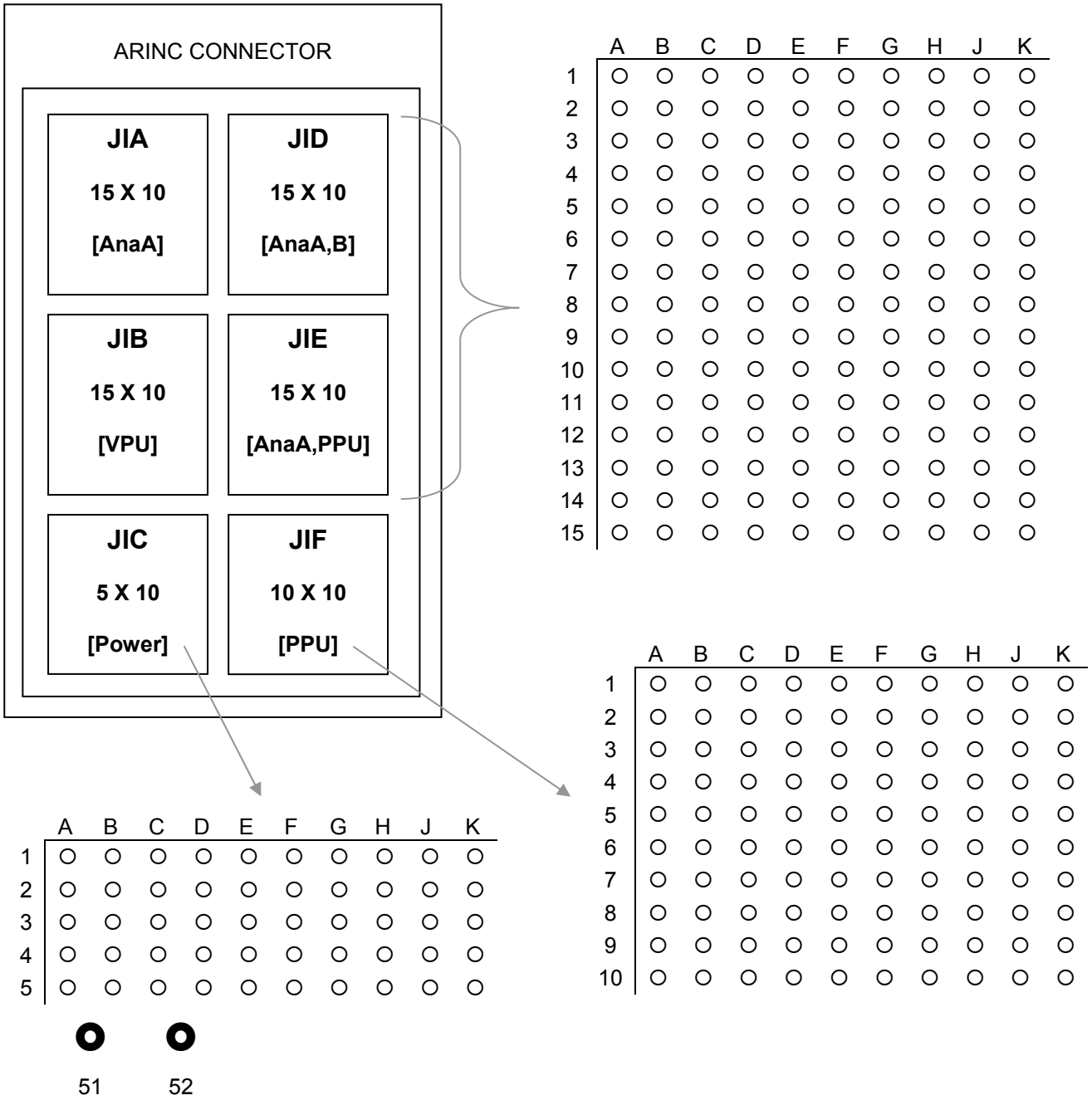


Figure A - 1 ARINC Connector & Pin Outs

JIA	A	B	C	D	E
1	USER DEFINED SIGNAL DCMIDA_1+	USER DEFINED SIGNAL DCMIDA_1-	USER DEFINED SIGNAL_DCLOWA_11+	USER DEFINED SIGNAL_DCLOWA_11-	USER DEFINED SIGNAL_DCMIDA_9+
2	USER DEFINED SIGNAL_DCMIDA_12+	USER DEFINED SIGNAL_DCMIDA_12-	USER DEFINED SIGNAL_DCMIDA_2+	USER DEFINED SIGNAL_DCMIDA_2-	USER DEFINED SIGNAL_DCLOWA_8+
3	USER DEFINED SIGNAL_DCLOWA_5+	USER DEFINED SIGNAL_DCLOWA_5-	USER DEFINED SIGNAL_DCLOWA_1+	USER DEFINED SIGNAL_DCLOWA_1-	USER DEFINED SIGNAL_DCLOWA_13+
4	USER DEFINED SIGNAL_DCLOWA_10+	USER DEFINED SIGNAL_DCLOWA_10-	USER DEFINED SIGNAL_DCMIDA_7+	USER DEFINED SIGNAL_DCMIDA_7-	USER DEFINED SIGNAL_DCMIDA_4+
5	USER DEFINED SIGNAL_DCHIA_2+	USER DEFINED SIGNAL_DCHIA_2-	USER DEFINED SIGNAL_DCMIDA_6+	USER DEFINED SIGNAL_DCMIDA_6-	USER DEFINED SIGNAL_DCMIDA_10+
6	USER DEFINED SIGNAL_DCLOWA_6+	USER DEFINED SIGNAL_DCLOWA_6-	USER DEFINED SIGNAL_DCMIDA_14+	USER DEFINED SIGNAL_DCMIDA_14-	USER DEFINED SIGNAL_DCHIA_1+
7	CGND	CGND	CGND	CGND	CGND
8	USER DEFINED SIGNAL_ACCoupA_1c+	USER DEFINED SIGNAL_ACCoupA_1c-	USER DEFINED SIGNAL_ACCoupA_3c+	USER DEFINED SIGNAL_ACCoupA_3c-	USER DEFINED SIGNAL_DisA_5
9	USER DEFINED SIGNAL_ACCoupA_1b+	USER DEFINED SIGNAL_ACCoupA_1b-	USER DEFINED SIGNAL_ACCoupA_3b+	USER DEFINED SIGNAL_ACCoupA_3b-	USER DEFINED SIGNAL_DisA_46
10	USER DEFINED SIGNAL_ACCoupA_1a+	USER DEFINED SIGNAL_ACCoupA_1a-	USER DEFINED SIGNAL_ACCoupA_2c+	USER DEFINED SIGNAL_ACCoupA_2c-	USER DEFINED SIGNAL_DisA_11
11	USER DEFINED SIGNAL_ACCoupA_2a+	USER DEFINED SIGNAL_ACCoupA_2a-	USER DEFINED SIGNAL_ACCoupA_4c+	USER DEFINED SIGNAL_ACCoupA_4c-	USER DEFINED SIGNAL_DisA_6
12	USER DEFINED SIGNAL_ACCoupA_2d+	USER DEFINED SIGNAL_ACCoupA_2d-	USER DEFINED SIGNAL_ACCoupA_4b+	USER DEFINED SIGNAL_ACCoupA_4b-	USER DEFINED SIGNAL_DisA_18
13	USER DEFINED SIGNAL_ACCoupA_1d+	USER DEFINED SIGNAL_ACCoupA_1d-	USER DEFINED SIGNAL_ACCoupA_4a+	USER DEFINED SIGNAL_ACCoupA_4a-	USER DEFINED SIGNAL_DisA_17
14	USER DEFINED SIGNAL_ACCoupA_3a+	USER DEFINED SIGNAL_ACCoupA_3a-	USER DEFINED SIGNAL_ACCoupA_4d+	USER DEFINED SIGNAL_ACCoupA_4d-	USER DEFINED SIGNAL_DisA_12
15	USER DEFINED SIGNAL_ACCoupA_3d+	USER DEFINED SIGNAL_ACCoupA_3d-	USER DEFINED SIGNAL_ACCoupA_2b+	USER DEFINED SIGNAL_ACCoupA_2b-	USER DEFINED SIGNAL_DisA_35

JIA	F	G	H	J	K
1	USER DEFINED SIGNAL_DCMIDA_9-	USER DEFINED SIGNAL_DCMIDA_5+	USER DEFINED SIGNAL_DCMIDA_5-	USER DEFINED SIGNAL_DCMIDA_8+	USER DEFINED SIGNAL_DCMIDA_8-
2	USER DEFINED SIGNAL_DCLOWA_8-	USER DEFINED SIGNAL_DCLOWA_4+	USER DEFINED SIGNAL_DCLOWA_4-	USER DEFINED SIGNAL_DCMIDA_13+	USER DEFINED SIGNAL_DCMIDA_13-
3	USER DEFINED SIGNAL_DCLOWA_13-	USER DEFINED SIGNAL_DCLOWA_9+	USER DEFINED SIGNAL_DCLOWA_9-	USER DEFINED SIGNAL_DCLOWA_2+	USER DEFINED SIGNAL_DCLOWA_2-
4	USER DEFINED SIGNAL_DCMIDA_4-	USER DEFINED SIGNAL_DCLOWA_14+	USER DEFINED SIGNAL_DCLOWA_14-	USER DEFINED SIGNAL_DCLOWA_7+	USER DEFINED SIGNAL_DCLOWA_7-
5	USER DEFINED SIGNAL_DCMIDA_10-	USER DEFINED SIGNAL_DCLOWA_3+	USER DEFINED SIGNAL_DCLOWA_3-	USER DEFINED SIGNAL_DCLOWA_12+	USER DEFINED SIGNAL_DCLOWA_12-
6	USER DEFINED SIGNAL_DCHIA_1-	USER DEFINED SIGNAL_DCMIDA_11+	USER DEFINED SIGNAL_DCMIDA_11-	USER DEFINED SIGNAL_DCMIDA_3+	USER DEFINED SIGNAL_DCMIDA_3-
7	CGND	CGND	CGND	CGND	CGND
8	USER DEFINED SIGNAL_DisA_30	USER DEFINED SIGNAL_DisA_1	USER DEFINED SIGNAL_DisA_25	USER DEFINED SIGNAL_DisA_3	USER DEFINED SIGNAL_DisA_27
9	USER DEFINED SIGNAL_DisA_29	USER DEFINED SIGNAL_DisA_8	USER DEFINED SIGNAL_DisA_32	USER DEFINED SIGNAL_DisA_10	USER DEFINED SIGNAL_DisA_34
10	USER DEFINED SIGNAL_DisA_36	USER DEFINED SIGNAL_DisA_19	USER DEFINED SIGNAL_DisA_43	USER DEFINED SIGNAL_DisA_21	USER DEFINED SIGNAL_DisA_45
11	USER DEFINED SIGNAL_DisA_47	USER DEFINED SIGNAL_DisA_14	USER DEFINED SIGNAL_DisA_38	USER DEFINED SIGNAL_DisA_16	USER DEFINED SIGNAL_DiscA_40
12	USER DEFINED SIGNAL_DisA_42	USER DEFINED SIGNAL_DisA_13	USER DEFINED SIGNAL_DisA_37	USER DEFINED SIGNAL_DisA_15	USER DEFINED SIGNAL_DisA_39
13	USER DEFINED SIGNAL_DisA_41	USER DEFINED SIGNAL_DisA_48	USER DEFINED SIGNAL_DisA_20	USER DEFINED SIGNAL_DisA_44	USER DEFINED SIGNAL_DisA_22
14	USER DEFINED SIGNAL_DisA_24	USER DEFINED SIGNAL_DisA_31	USER DEFINED SIGNAL_DisA_7	USER DEFINED SIGNAL_DisA_33	USER DEFINED SIGNAL_DisA_9
15	USER DEFINED SIGNAL_DisA_23	USER DEFINED SIGNAL_DisA_26	USER DEFINED SIGNAL_DisA_4	USER DEFINED SIGNAL_DisA_28	USER DEFINED SIGNAL_DisA_2

Table A - 1 ARINC Connector J1A (15 x 10) [AnaA]

JIB	A	B	C	D	E
1	A_MR01_H	A_MR01_L	A_MR02_H	A_MR02_L	A_MR03_H
2	A_MR06_L	A_MR06_H	A_TR01_L	A_TR01_H	A_TR02_L
3	CGND	CGND	CGND	CGND	CGND
4	A_DT13_H	A_DT13_L	A_DT14_H	A_DT14_L	A_DT15_H
5	A_DT17_L	A_DT17_H	A_DT16_L	A_DT16_H	A_DT15_L
6	A_DT18_H	A_DT18_L	A_DT19_H	A_DT19_L	A_DT20_H
7	A_DT21_L	A_DT21_H	A_DT22_L	A_DT22_H	A_DT23_L
8	A_DT26_H	A_DT26_L	A_DT27_H	A_DT27_L	A_DT28_H
9	A_DT31_L	A_DT31_H	A_DT32_L	A_DT32_H	A_DT1_L
10	A_DT4_H	A_DT4_L	A_DT5_H	A_DT5_L	A_DT6_H
11	A_DT9_L	A_DT9_H	A_DT10_L	A_DT10_H	A_DT11_L
12	CGND	CGND	CGND	CGND	CGND
13	INDX_TRO_L	INDX_TRO_H	INDX_OPT_L	Frequency_10+, [T_SPA1_H]	Frequency_07-, [T_ENG3_L]
14	INDX_MRO_H	INDX_MRO_L	INDX_OPT_H	Frequency_10-, [T_SPA1_L]	Frequency_06+, [T_ENG2_H]
15	INDX_SH_L	INDX_SH_H	Frequency_11-, [T_SPA2_L]	Frequency_11+, [T_SPA2_H]	Frequency_05-, [T_ENG1_L]

JIB	F	G	H	J	K
1	A_MR03_L	A_MR04_H	A_MR04_L	A_MR05_L	A_MR05_H
2	A_TR02_H	CGND	CGND	CGND	CGND
3	CGND	Sensor GND	A_ENG1A_H	A_ENG1_L	A_ENG1B_H
4	CGND	A_ENG2A_H	A_ENG2_L	A_ENG2B_H	Sensor GND
5	CGND	Sensor GND	A_ENG3A_H	A_ENG3_L	A_ENG3B_H
6	A_DT20_L	CGND	CGND	CGND	CGND
7	A_DT23_H	A_DT24_L	A_DT24_H	A_DT25_L	A_DT25_H
8	A_DT28_L	A_DT29_H	A_DT29_L	A_DT30_H	A_DT30_L
9	A_DT1_H	A_DT2_L	A_DT2_H	A_DT3_L	A_DT3_H
10	A_DT6_L	A_DT7_H	A_DT7_L	A_DT8_H	A_DT8_L
11	A_DT11_H	A_DT12_L	A_DT12_H	CGND	CGND
12	CGND	CGND	CGND	MIC_L	MIC_H
13	Frequency_07+, [T_ENG3_H]	Frequency_09-, [T_DT2_L]	Frequency_09+, [T_DT2_H]	CGND	CGND
14	Frequency_06-, [T_ENG2_L]	Frequency_08+, [T_DT1_H]	CGND	OPT_TRK1_L	OPT_TRK1_H
15	Frequency_05+, [T_ENG1_H]	Frequency_08- [T_DT1_L]	CGND	OPT_TRK2_H	OPT_TRK2_L

Table A - 2 ARINC Connector J1B (15 x 10) [VPU]

JIC	A	B	C	D	E
1	JIC	A	B	C	D
2	1	T_SPA3_H	T_SPA3_L	CGND	Not Used
3	2	USER DEFINED SIGNAL_DISOUTB_1+	USER DEFINED SIGNAL_DISOUTB_1-	+28V_Out2 (1), [+28V_RDC (1)]	+28V_Out2_Rtn (1), [+28V_RDC_Rtn (1)]
4	3	USER DEFINED SIGNAL_ENSHUB_1+	USER DEFINED SIGNAL_GENSHUB_1-	USER DEFINED SIGNAL_DISOUTB_2+	DISOUTB_2-
5	4	USER DEFINED SIGNAL_DCHIB_1+	USER DEFINED SIGNAL_DCHIB_1-	USER DEFINED SIGNAL_DCHIB_2+	USER DEFINED SIGNAL_DCHIB_2-
	+28V_Power		+28V_Power_Rtn		
	51		52		

JIC	F	G	H	J	K
1	Not Used	Not Used	CGND	+12V_Trk (1)	+12V_Trk_Rtn (1)
2	+28V_Out2_Rtn (2), [+28V_RDC_Rtn (2)]	USER DEFINED SIGNAL_DISOUTA_4+	USER DEFINED SIGNAL_DISOUTA_4-	Trk_Lit+ (1)	Trk_Lit- (1)
3	USER DEFINED SIGNAL_DISOUTA_3-	USER DEFINED SIGNAL_DISOUTA_1+	USER DEFINED SIGNAL_DISOUTA_1-	+28V_Out1, [+28V_DTU]	+28V_Out1_Rtn, [+28V_DTU_Rtn]
4	USER DEFINED SIGNAL_DISOUTB_3-	HUMS_Fail_Disc+	HUMS_Fail_Disc-	Trk_Lit+ (2)	Trk_Lit- (2)
5	USER DEFINED SIGNAL_DISOUTA_2-	+12V_Trk (2)	+12V_Trk_Rtn (2)	USER DEFINED SIGNAL_DISOUTB_4+	USER DEFINED SIGNAL_DISOUTB_4-

Table A - 3 ARINC Connector J1C (5 x 10) [Power Supply]

JID	A	B	C	D	E
1	USER DEFINED SIGNAL_GENSHUA_2+	USER DEFINED SIGNAL_GENSHUA_2-	USER DEFINED SIGNAL_+10VExcA_2+	USER DEFINED SIGNAL_+10VExcA_2-	USER DEFINED SIGNAL_DCHIA_3+
2	USER DEFINED SIGNAL_DCHIA_4+	USER DEFINED SIGNAL_DCHIA_4-	USER DEFINED SIGNAL_IOatPrbA_2+	USER DEFINED SIGNAL_IOatPrbA_2-	USER DEFINED SIGNAL_IDCExcA_3+
3	USER DEFINED SIGNAL_DCCoupA_1d+	USER DEFINED SIGNAL_DCCoupA_1d-	USER DEFINED SIGNAL_IDCExcA_4+	USER DEFINED SIGNAL_IDCExcA_4-	USER DEFINED SIGNAL_DCCoupA_1a+
4	USER DEFINED SIGNAL_+5VExcA_2+	USER DEFINED SIGNAL_+5VExcA_2-	USER DEFINED SIGNAL_HDLA+	USER DEFINED SIGNAL_HDLA-	CGND
5	USER DEFINED SIGNAL_+5VExcA_1+	USER DEFINED SIGNAL_+5VExcA_1-	USER DEFINED SIGNAL_+10VExcA_1+	USER DEFINED SIGNAL_+10VExcA_1-	CGND
6	USER DEFINED SIGNAL_IOatPrbA_1+	USER DEFINED SIGNAL_IOatPrbA_1-	USER DEFINED SIGNAL_IDCExcA_1+	USER DEFINED SIGNAL_IDCExcA_1-	USER DEFINED SIGNAL_ACExcA_1+
7	CGND	CGND	CGND	CGND	CGND
8	USER DEFINED SIGNAL_ACCoupB_1c+	USER DEFINED SIGNAL_ACCoupB_1c-	USER DEFINED SIGNAL_ACCoupB_2d+	USER DEFINED SIGNAL_ACCoupB_2d-	USER DEFINED SIGNAL_DisB_26
9	USER DEFINED SIGNAL_ACCoupB_1b+	USER DEFINED SIGNAL_ACCoupB_1b-	USER DEFINED SIGNAL_ACCoupB_4a+	USER DEFINED SIGNAL_ACCoupB_4a-	USER DEFINED SIGNAL_DisB_37
10	USER DEFINED SIGNAL_ACCoupB_1d+	USER DEFINED SIGNAL_ACCoupB_1d-	USER DEFINED SIGNAL_ACCoupB_3d+	USER DEFINED SIGNAL_ACCoupB_3d-	USER DEFINED SIGNAL_DisB_44
11	USER DEFINED SIGNAL_ACCoupB_2c+	USER DEFINED SIGNAL_ACCoupB_2c-	USER DEFINED SIGNAL_ACCoupB_3c+	USER DEFINED SIGNAL_ACCoupB_3c-	USER DEFINED SIGNAL_DisB_43
12	USER DEFINED SIGNAL_ACCoupB_2b+	USER DEFINED SIGNAL_ACCoupB_2b-	USER DEFINED SIGNAL_ACCoupB_4b+	USER DEFINED SIGNAL_ACCoupB_4b-	USER DEFINED SIGNAL_DisB_38
13	USER DEFINED SIGNAL_ACCoupB_2a+	USER DEFINED SIGNAL_ACCoupB_2a-	USER DEFINED SIGNAL_ACCoupB_4d+	USER DEFINED SIGNAL_ACCoupB_4d-	USER DEFINED SIGNAL_DisB_25
14	USER DEFINED SIGNAL_ACCoupB_1a+	USER DEFINED SIGNAL_ACCoupB_1a-	USER DEFINED SIGNAL_ACCoupB_4c+	USER DEFINED SIGNAL_ACCoupB_4c-	USER DEFINED SIGNAL_DisB_10
15	USER DEFINED SIGNAL_ACCoupB_3a+	USER DEFINED SIGNAL_ACCoupB_3a-	USER DEFINED SIGNAL_ACCoupB_3b+	USER DEFINED SIGNAL_ACCoupB_3b-	USER DEFINED SIGNAL_DisB_9

JID	F	G	H	J	K
1	USER DEFINED SIGNAL_DCHIA_3-	USER DEFINED SIGNAL_GENSHUA_1+	USER DEFINED SIGNAL_GENSHUA_1-	USER DEFINED SIGNAL_DCCoupA_1b+	USER DEFINED SIGNAL_DCCoupA_1b-
2	USER DEFINED SIGNAL_IDCExcA_3-	USER DEFINED SIGNAL_IDCExcA_2+	USER DEFINED SIGNAL_IDCExcA_2-	USER DEFINED SIGNAL_DCCoupA_1c+	USER DEFINED SIGNAL_DCCoupA_1c-
3	USER DEFINED SIGNAL_DCCoupA_1a-	USER DEFINED SIGNAL_Bit +28V Reg	USER DEFINED SIGNAL_Bit +28V Reg Rtn	CGND	CGND
4	CGND	USER DEFINED SIGNAL_ACExcA_2+	USER DEFINED SIGNAL_ACExcA_2-	USER DEFINED SIGNAL_FreqA_2+	USER DEFINED SIGNAL_FreqA_2-
5	CGND	USER DEFINED SIGNAL_FreqA_3+	USER DEFINED SIGNAL_FreqA_3-	USER DEFINED SIGNAL_FreqA_5+	USER DEFINED SIGNAL_FreqA_5-
6	USER DEFINED SIGNAL_ACExcA_1-	USER DEFINED SIGNAL_FreqA_1+	USER DEFINED SIGNAL_FreqA_1-	USER DEFINED SIGNAL_FreqA_4+	USER DEFINED SIGNAL_FreqA_4-
7	CGND	CGND	CGND	CGND	CGND
8	USER DEFINED SIGNAL_DisB_4	USER DEFINED SIGNAL_DisB_28	USER DEFINED SIGNAL_DisB_6	USER DEFINED SIGNAL_DisB_30	USER DEFINED SIGNAL_DisB_2
9	USER DEFINED SIGNAL_DiscB_15	USER DEFINED SIGNAL_DisB_27	USER DEFINED SIGNAL_DisB_17	USER DEFINED SIGNAL_DisB_41	USER DEFINED SIGNAL_DisB_13
10	USER DEFINED SIGNAL_DisB_22	USER DEFINED SIGNAL_DisB_34	USER DEFINED SIGNAL_DisB_24	USER DEFINED SIGNAL_DisB_48	USER DEFINED SIGNAL_DisB_20
11	USER DEFINED SIGNAL_DisB_21	USER DEFINED SIGNAL_DisB_31	USER DEFINED SIGNAL_DisB_23	USER DEFINED SIGNAL_DisB_47	USER DEFINED SIGNAL_DisB_19
12	USER DEFINED SIGNAL_DisB_16	USER DEFINED SIGNAL_DisB_40	USER DEFINED SIGNAL_DisB_18	USER DEFINED SIGNAL_DisB_42	USER DEFINED SIGNAL_DisB_14
13	USER DEFINED SIGNAL_DisB_3	USER DEFINED SIGNAL_DisB_39	USER DEFINED SIGNAL_DisB_5	USER DEFINED SIGNAL_DisB_29	USER DEFINED SIGNAL_DisB_1
14	USER DEFINED SIGNAL_DisB_32	USER DEFINED SIGNAL_DisB_12	USER DEFINED SIGNAL_DisB_46	USER DEFINED SIGNAL_DisB_8	USER DEFINED SIGNAL_DisB_36
15	USER DEFINED SIGNAL_DisB_33	USER DEFINED SIGNAL_DisB_11	USER DEFINED SIGNAL_DisB_35	USER DEFINED SIGNAL_DisB_7	USER DEFINED SIGNAL_DisB_45

Table A - 4 ARINC Connector J1D (15 x 10) [AnaA,B]

JIE	A	B	C	D	E
1	USER DEFINED SIGNAL_IDCExcB_2+	USER DEFINED SIGNAL_IDCExcB_2-	USER DEFINED SIGNAL_DCMIDB_2+	USER DEFINED SIGNAL_DCMIDB_2-	USER DEFINED SIGNAL_+10VExcB_1+
2	USER DEFINED SIGNAL_DCLOWB_10+	USER DEFINED SIGNAL_DCLOWB_10-	USER DEFINED SIGNAL_IOAtPrbB_1+	USER DEFINED SIGNAL_IOAtPrbB_1-	USER DEFINED SIGNAL_DCLOWB_11+
3	USER DEFINED SIGNAL_DCLOWB_9+	USER DEFINED SIGNAL_DCLOWB_9-	USER DEFINED SIGNAL_DCMIDB_12+	USER DEFINED SIGNAL_DCMIDB_12-	USER DEFINED SIGNAL_IOAtPrbB_2+
4	USER DEFINED SIGNAL_DCLOWB_4+	USER DEFINED SIGNAL_DCLOWB_4-	USER DEFINED SIGNAL_DCMIDB_7+	USER DEFINED SIGNAL_DCMIDB_7-	USER DEFINED SIGNAL_DCMIDB_13+
5	USER DEFINED SIGNAL_DCLOWB_14+	USER DEFINED SIGNAL_DCLOWB_14-	USER DEFINED SIGNAL_DCCoupB_1b+	USER DEFINED SIGNAL_DCCoupB_1b-	USER DEFINED SIGNAL_+5VExcB_2+
6	USER DEFINED SIGNAL_DCHIB_3+	USER DEFINED SIGNAL_DCHIB_3-	USER DEFINED SIGNAL_DCCoupB_1a+	USER DEFINED SIGNAL_DCCoupB_1a-	USER DEFINED SIGNAL_DCLOWB_3+
7	USER DEFINED SIGNAL_DCMIDB_10+	USER DEFINED SIGNAL_DCMIDB_10-	USER DEFINED SIGNAL_+5VExcB_1+	USER DEFINED SIGNAL_+5VExcB_1-	USER DEFINED SIGNAL_DCCoupB_1d+
8	USER DEFINED SIGNAL_DCMIDB_5+	USER DEFINED SIGNAL_DCMIDB_5-	USER DEFINED SIGNAL_DCLOWB_2+	USER DEFINED SIGNAL_DCLOWB_2-	DGND [6]
9	USER DEFINED SIGNAL_HDLB+	USER DEFINED SIGNAL_HDLB-	USER DEFINED SIGNAL_DCMIDB_3+	USER DEFINED SIGNAL_DCMIDB_3-	Bit -15VA
10	USER DEFINED SIGNAL_DCLOWB_5+	USER DEFINED SIGNAL_DCLOWB_5-	USER DEFINED SIGNAL_DCLOWB_12+	USER DEFINED SIGNAL_DCLOWB_12-	Bit +15VA
11	USER DEFINED SIGNAL_DCLOWB_1+	USER DEFINED SIGNAL_DCLOWB_1-	USER DEFINED SIGNAL_DCLOWB_7+	USER DEFINED SIGNAL_DCLOWB_7-	BIT+12V
12	Bit +2.9VB	BIT+5V	Bit +3.3V	Bit +2.5V	Bit-12V
13	1553 RT0	1553 RT5	1553 RT4	1553 RT1	1553 RT3
14	DISCRETE (12)	DISCRETE (0)	DISCRETE (2)	DISCRETE (8)	DISCRETE (4)
15	DISCRETE (1)	DISCRETE (3)	DISCRETE (11)	DGND [6]	USER DEFINED SIGNAL_FreqB_3+

JIE	F	G	H	J	K
1	USER DEFINED SIGNAL_+10VExcB_1-	USER DEFINED SIGNAL_DCCoupB_1c+	USER DEFINED SIGNAL_DCCoupB_1c-	USER DEFINED SIGNAL_DCLOWB_6+	USER DEFINED SIGNAL_DCLOWB_6-
2	USER DEFINED SIGNAL_DCLOWB_11-	USER DEFINED SIGNAL_DCMIDB_8+	USER DEFINED SIGNAL_DCMIDB_8-	USER DEFINED SIGNAL_IDCExcB_3+	USER DEFINED SIGNAL_IDCExcB_3-
3	USER DEFINED SIGNAL_IOAtPrbB_2-	USER DEFINED SIGNAL_DCMIDB_9+	USER DEFINED SIGNAL_DCMIDB_9-	USER DEFINED SIGNAL_DCMIDB_1+	USER DEFINED SIGNAL_DCMIDB_1-
4	USER DEFINED SIGNAL_DCMIDB_13-	USER DEFINED SIGNAL_DCMIDB_4+	USER DEFINED SIGNAL_DCMIDB_4-	USER DEFINED SIGNAL_DCHIB_4+	USER DEFINED SIGNAL_DCHIB_4-
5	USER DEFINED SIGNAL_+5VExcB_2-	USER DEFINED SIGNAL_DCLOWB_13+	USER DEFINED SIGNAL_DCLOWB_13-	USER DEFINED SIGNAL_DCMIDB_11+	USER DEFINED SIGNAL_DCMIDB_11-
6	USER DEFINED SIGNAL_DCLOWB_3-	USER DEFINED SIGNAL_DCMIDB_14+	USER DEFINED SIGNAL_DCMIDB_14-	USER DEFINED SIGNAL_DCMIDB_6+	USER DEFINED SIGNAL_DCMIDB_6-
7	USER DEFINED SIGNAL_DCCoupB_1d-	USER DEFINED SIGNAL_DCLOWB_8+	USER DEFINED SIGNAL_DCLOWB_8-	USER DEFINED SIGNAL_IDCExcB_1+	USER DEFINED SIGNAL_IDCExcB_1-
8	Bit Battery+	USER DEFINED SIGNAL_+10VExcB_2+	USER DEFINED SIGNAL_+10VExcB_2-	USER DEFINED SIGNAL_IDCExcB_4+	USER DEFINED SIGNAL_IDCExcB_4-
9	Bit Vbat	AGND [T, 6]	Bit -15VB	Bit Sensor GND	Bit +28V VPU
10	Bit Battery-	USER DEFINED SIGNAL_FreqB_1+	USER DEFINED SIGNAL_FreqB_1-	USER DEFINED SIGNAL_FreqB_4+	USER DEFINED SIGNAL_FreqB_4-
11	Bit +2.9VA	Frequency_04+	Frequency_04-	USER DEFINED SIGNAL_FreqB_2+	USER DEFINED SIGNAL_FreqB_2-
12	Bit +15VB	USER DEFINED SIGNAL_ACExcB_2+	USER DEFINED SIGNAL_ACExcB_2-	USER DEFINED SIGNAL_FreqB_5+	USER DEFINED SIGNAL_FreqB_5-
13	1553 RT2	DISCRETE (14)	DISCRETE (9)	USER DEFINED SIGNAL_ACExcB_1+	USER DEFINED SIGNAL_ACExcB_1-
14	DISCRETE (10)	DISCRETE (13)	DISCRETE (6)	DISCRETE (5)	DISCRETE (7)
15	USER DEFINED SIGNAL_FreqB_3-	A717_Tx+2	A717_Tx-2	A717_Tx+1	A717_Tx-1

Table A - 5 ARINC Connector J1E (15 x 10) [AnaA,PPU]

JIF	A	B	C	D	E
1	Frequency_00+	Frequency_00-	Frequency_03+	Frequency_03-	+A429_TX00
2	Frequency_02+	Frequency_02-	Frequency_01+	Frequency_01-	422_TX+3
3	+A429_TX03	-A429_TX03	+A429_RX03	-A429_RX03	422_RX+5\485+1
4	422_TX+5	422_TX-5	422_TX+6	422_TX-6	+A429_RX04
5	422_RX+1	422_RX-1\232_Rx1	422_RX+6\485+2	422_RX-6\485-2	+A429_TX01
6	+A429_RX13	-A429_RX13	+A429_RX09	-A429_RX09	422_TX+4
7	+A429_RX08	-A429_RX08	422_CTS_+3	422_CTS_-3	422_RTS_+3
8	+A429_RX12	-A429_RX12	422_RX+2	422_RX-2/232_Rx2	422_RX+3
9	CGND	CGND	CGND	CGND	CGND
10	1553A(Bus B)+	1553A(Bus B)-	1553BTCP(Bus B)+	1553BTCP(Bus B)-	1553A(Bus A)+

JIF	F	G	H	J	K
1	-A429_TX00	422_RX+7\485+3	422_RX-7\485-3	422_TX+7	422_TX-7
2	422_TX-3\232_Tx3	+A429_RX00	-A429_RX00	422_RX+4\485+0	422_RX-4\485-0
3	422_RX-5\485-1	+A429_RX10	-A429_RX10	422_TX+1	422_TX-1\232_Tx1
4	-A429_RX04	+A429_RX02	-A429_RX02	+A429_RX07	-A429_RX07
5	-A429_TX01	+A429_RX11	-A429_RX11	+A429_TX02	-A429_TX02
6	422_TX-4	+A429_RX06	-A429_RX06	+A429_RX01	-A429_RX01
7	422_RTS_-3	+A429_RX05	-A429_RX05	422_TX+2	422_TX-2/232_Tx2
8	422_RX-3/232_Rx3	CGND	CGND	CGND	CGND
9	CGND	CGND	CGND	1553BTCP(Bus A)+	1553BTCP(Bus A)-
10	1553A(Bus A)-	1553BDCP(Bus B)+	1553BDCP(Bus B)-	1553BDCP(Bus A)+	1553BDCP(Bus A)-

Table A - 6 ARINC Connector J1F (10 x 10) [PPU]

Appendix B Spare Board Signal Assignment & Geometry

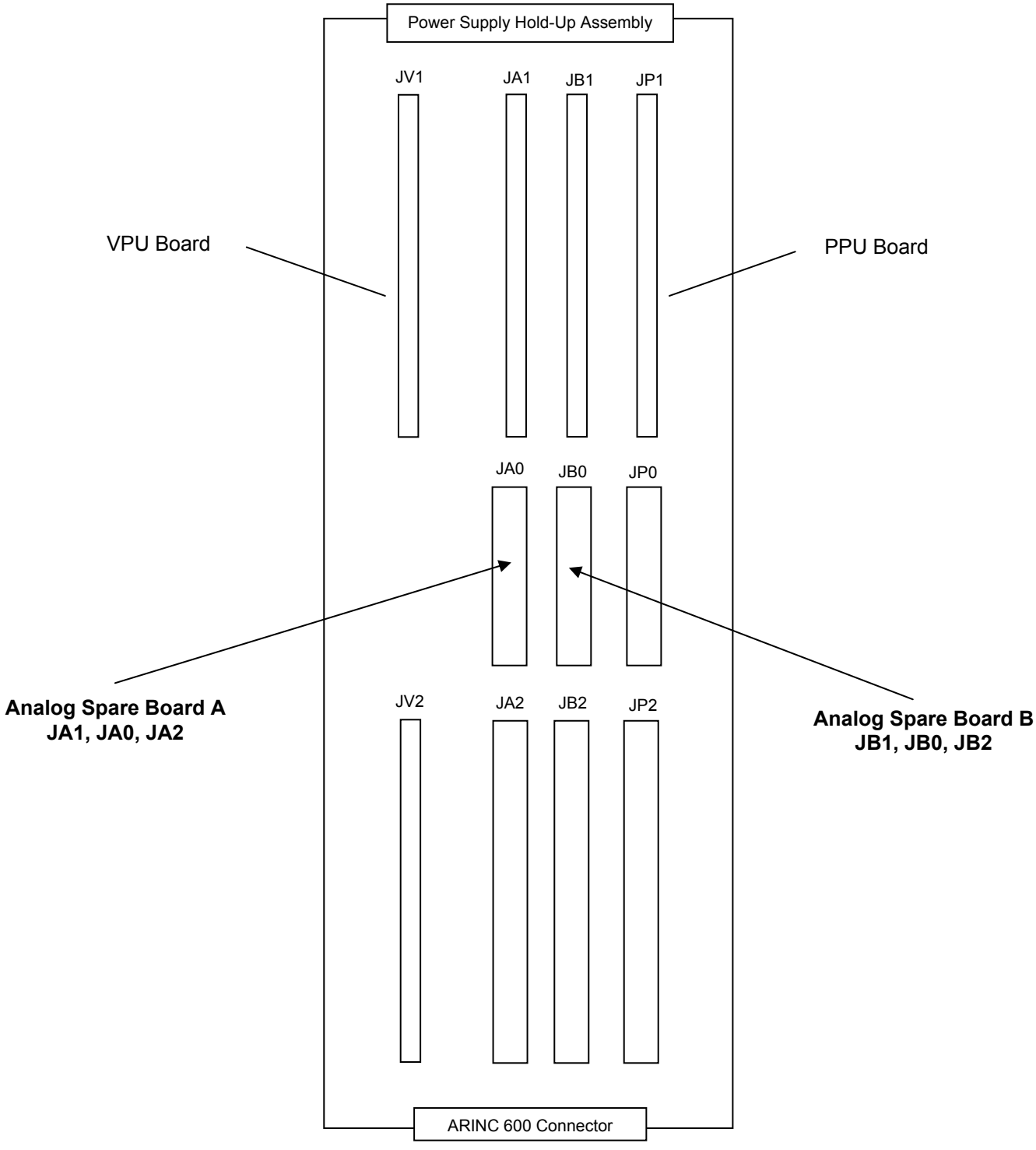


Figure B - 1 MPU Backplane Block Diagram

JA1

PIN	ROW A	ROW B	ROW C
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	DGND	BG2OUT*	DGND
10	SYSCLK	BG3IN*	SYSFAIL*
11	DGND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	DGND	BR3*	A23
16	DTACK*	AM0	A22
17	DGND	AM1	A21
18	AS*	AM2	A20
19	DGND	AM3	A19
20	IACK*	DGND	A18
21	IACKIN*	SERCLK	A17
22	IACKOUT*	SERDAT*	A16
23	AM4	DGND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	NC (DGnd)	+12V
32	+5V	+5V	+5V

Table B - 1 Spare Board A, JA1 Pin Outs

JA0

PIN	ROW A	ROW B	ROW C	ROW D	ROW E
1	USER DEFINED SIGNAL_ACCoupA_1a+	USER DEFINED SIGNAL_ACCoupA_1a-	AGND	USER DEFINED SIGNAL_DisA_22	USER DEFINED SIGNAL_DisA_41
2	USER DEFINED SIGNAL_ACCoupA_1b+	USER DEFINED SIGNAL_ACCoupA_1b-	USER DEFINED SIGNAL_DisA_7	USER DEFINED SIGNAL_DisA_23	USER DEFINED SIGNAL_DisA_42
3	USER DEFINED SIGNAL_ACCoupA_1c+	USER DEFINED SIGNAL_ACCoupA_1c-	USER DEFINED SIGNAL_DisA_8	USER DEFINED SIGNAL_DisA_24	USER DEFINED SIGNAL_DisA_43
4	USER DEFINED SIGNAL_ACCoupA_1d+	USER DEFINED SIGNAL_ACCoupA_1d-	USER DEFINED SIGNAL_DisA_9	USER DEFINED SIGNAL_DisA_25	USER DEFINED SIGNAL_DisA_44
5	USER DEFINED SIGNAL_ACCoupA_2a+	USER DEFINED SIGNAL_ACCoupA_2a-	USER DEFINED SIGNAL_DisA_10	USER DEFINED SIGNAL_DisA_26	USER DEFINED SIGNAL_DisA_45
6	USER DEFINED SIGNAL_ACCoupA_2b+	USER DEFINED SIGNAL_ACCoupA_2b-	USER DEFINED SIGNAL_DisA_11	USER DEFINED SIGNAL_DisA_27	USER DEFINED SIGNAL_DisA_46
7	USER DEFINED SIGNAL_ACCoupA_2c+	USER DEFINED SIGNAL_ACCoupA_2c-	USER DEFINED SIGNAL_DisA_12	USER DEFINED SIGNAL_DisA_28	USER DEFINED SIGNAL_DisA_47
8	USER DEFINED SIGNAL_ACCoupA_2d+	USER DEFINED SIGNAL_ACCoupA_2d-	USER DEFINED SIGNAL_DisA_13	USER DEFINED SIGNAL_DisA_29	USER DEFINED SIGNAL_DisA_48
9	USER DEFINED SIGNAL_ACCoupA_3a+	USER DEFINED SIGNAL_ACCoupA_3a-	USER DEFINED SIGNAL_DisA_14	USER DEFINED SIGNAL_DisA_30	USER DEFINED SIGNAL_FreqA_1+
10	USER DEFINED SIGNAL_ACCoupA_3b+	USER DEFINED SIGNAL_ACCoupA_3b-	USER DEFINED SIGNAL_DisA_15	USER DEFINED SIGNAL_DisA_31	USER DEFINED SIGNAL_FreqA_1-
11	USER DEFINED SIGNAL_ACCoupA_3c+	USER DEFINED SIGNAL_ACCoupA_3c-	USER DEFINED SIGNAL_DisA_16	USER DEFINED SIGNAL_DisA_32	USER DEFINED SIGNAL_FreqA_2+
12	USER DEFINED SIGNAL_ACCoupA_3d+	USER DEFINED SIGNAL_ACCoupA_3d-	USER DEFINED SIGNAL_DisA_17	USER DEFINED SIGNAL_DisA_33	USER DEFINED SIGNAL_FreqA_2-
13	USER DEFINED SIGNAL_ACCoupA_4a+	USER DEFINED SIGNAL_ACCoupA_4a-	USER DEFINED SIGNAL_DisA_18	USER DEFINED SIGNAL_DisA_34	USER DEFINED SIGNAL_FreqA_3+
14	USER DEFINED SIGNAL_ACCoupA_4b+	USER DEFINED SIGNAL_ACCoupA_4b-	USER DEFINED SIGNAL_DisA_19	USER DEFINED SIGNAL_DisA_35	USER DEFINED SIGNAL_FreqA_3-
15	USER DEFINED SIGNAL_ACCoupA_4c+	USER DEFINED SIGNAL_ACCoupA_4c-	USER DEFINED SIGNAL_DisA_20	USER DEFINED SIGNAL_DisA_36	USER DEFINED SIGNAL_FreqA_4+
16	USER DEFINED SIGNAL_ACCoupA_4d+	USER DEFINED SIGNAL_ACCoupA_4d-	USER DEFINED SIGNAL_DisA_21	USER DEFINED SIGNAL_DisA_37	USER DEFINED SIGNAL_FreqA_4-

17	USER DEFINED SIGNAL_DisA_1	USER DEFINED SIGNAL_DisA_4	+28V Reg	USER DEFINED SIGNAL_DisA_38	USER DEFINED SIGNAL_FreqA_5+
18	USER DEFINED SIGNAL_DisA_2	USER DEFINED SIGNAL_DisA_5	+28V Reg Rtn	USER DEFINED SIGNAL_DisA_39	USER DEFINED SIGNAL_FreqA_5-
19	USER DEFINED SIGNAL_DisA_3	USER DEFINED SIGNAL_DisA_6	AGND	USER DEFINED SIGNAL_DisA_40	USER DEFINED SIGNAL_A/B Discrete

Table B - 2 Spare Board A, JA0 Pin Outs

JA2

PIN	ROW Z	ROW A	ROW B	ROW C	ROW D
1	USER DEFINED SIGNAL_DCLOW A_1+	USER DEFINED SIGNAL_DCLOW A_1-	+5V	Low Hold Up	AGND
2	USER DEFINED SIGNAL_DCLOW A_2+	USER DEFINED SIGNAL_DCLOW A_2-	DGND	USER DEFINED SIGNAL_IOatPrA_1+	USER DEFINED SIGNAL_IOatPrA_1-
3	USER DEFINED SIGNAL_DCLOW A_3+	USER DEFINED SIGNAL_DCLOW A_3-	RESERVED	USER DEFINED SIGNAL_IOtPrbA_2+	USER DEFINED SIGNAL_IOtPrbA_2-
4	USER DEFINED SIGNAL_DCLOW A_4+	USER DEFINED SIGNAL_DCLOW A_4-	A24	USER DEFINED SIGNAL_IDCExcA_1+	USER DEFINED SIGNAL_IDCExcA_1-
5	USER DEFINED SIGNAL_DCLOW A_5+	USER DEFINED SIGNAL_DCLOW A_5-	A25	USER DEFINED SIGNAL_IDCExcA_2+	USER DEFINED SIGNAL_IDCExcA_2-
6	USER DEFINED SIGNAL_DCLOW A_6+	USER DEFINED SIGNAL_DCLOW A_6-	A26	USER DEFINED SIGNAL_IDCExcA_3+	USER DEFINED SIGNAL_IDCExcA_3-
7	USER DEFINED SIGNAL_DCLOW A_7+	USER DEFINED SIGNAL_DCLOW A_7-	A27	USER DEFINED SIGNAL_IDCExcA_4+	USER DEFINED SIGNAL_IDCExcA_4-
8	USER DEFINED SIGNAL_DCLOW A_8+	USER DEFINED SIGNAL_DCLOW A_8-	A28	Vbat	CJTA+
9	USER DEFINED SIGNAL_DCLOW A_9+	USER DEFINED SIGNAL_DCLOW A_9-	A29	USER DEFINED SIGNAL_DCCoupA_1a+	+15VA
10	USER DEFINED SIGNAL_DCLOW A_10+	USER DEFINED SIGNAL_DCLOW A_10-	A30	USER DEFINED SIGNAL_DCCoupA_1a-	-15VA
11	USER DEFINED SIGNAL_DCLOW A_11+	USER DEFINED SIGNAL_DCLOW A_11-	A31	USER DEFINED SIGNAL_DCCoupA_1b+	USER DEFINED SIGNAL_DCCoupA_1b-
12	USER DEFINED SIGNAL_DCLOW A_12+	USER DEFINED SIGNAL_DCLOW A_12-	DGND	USER DEFINED SIGNAL_DCCoupA_1c+	USER DEFINED SIGNAL_DCCoupA_1c-
13	USER DEFINED SIGNAL_DCLOW A_13+	USER DEFINED SIGNAL_DCLOW A_13-	+5V	USER DEFINED SIGNAL_DCCoupA_1d+	USER DEFINED SIGNAL_DCCoupA_1d-
14	USER DEFINED SIGNAL_DCLOW A_14+	USER DEFINED SIGNAL_DCLOW A_14-	D16	USER DEFINED SIGNAL_GENSHUA_1+	USER DEFINED SIGNAL_GENSHUA_1-
15	USER DEFINED SIGNAL_DCMIDA _1+	USER DEFINED SIGNAL_DCMIDA _1-	D17	USER DEFINED SIGNAL_GENSHUA_2+	USER DEFINED SIGNAL_GENSHUA_2-
16	USER DEFINED SIGNAL_DCMIDA _2+	USER DEFINED SIGNAL_DCMIDA _2-	D18	USER DEFINED SIGNAL_HDLA+	USER DEFINED SIGNAL_HDLA-

17	USER DEFINED SIGNAL_DCMIDA_3+	USER DEFINED SIGNAL_DCMIDA_3-	D19	USER DEFINED SIGNAL_DISOUTA_1+	USER DEFINED SIGNAL_DISOUTA_1-
18	USER DEFINED SIGNAL_DCMIDA_4+	USER DEFINED SIGNAL_DCMIDA_4-	D20	USER DEFINED SIGNAL_DISOUTA_2+	USER DEFINED SIGNAL_DISOUTA_2-
19	USER DEFINED SIGNAL_DCMIDA_5+	USER DEFINED SIGNAL_DCMIDA_5-	D21	USER DEFINED SIGNAL_DISOUTA_3+	USER DEFINED SIGNAL_DISOUTA_3-
20	USER DEFINED SIGNAL_DCMIDA_6+	USER DEFINED SIGNAL_DCMIDA_6-	D22	USER DEFINED SIGNAL_DISOUTA_4+	USER DEFINED SIGNAL_DISOUTA_4-
21	USER DEFINED SIGNAL_DCMIDA_7+	USER DEFINED SIGNAL_DCMIDA_7-	D23	USER DEFINED SIGNAL_ACExcA_1+	USER DEFINED SIGNAL_ACExcA_1-
22	USER DEFINED SIGNAL_DCMIDA_8+	USER DEFINED SIGNAL_DCMIDA_8-	DGND	+A429_TX00 (RESERVED)	USER DEFINED SIGNAL_ACExcA_2+
23	USER DEFINED SIGNAL_DCMIDA_9+	USER DEFINED SIGNAL_DCMIDA_9-	D24	-A429_TX00 (RESERVED)	USER DEFINED SIGNAL_ACExcA_2-
24	USER DEFINED SIGNAL_DCMIDA_10+	USER DEFINED SIGNAL_DCMIDA_10-	D25	+A429_RX00 (RESERVED)	USER DEFINED SIGNAL_+5VExcA_1+
25	USER DEFINED SIGNAL_DCMIDA_11+	USER DEFINED SIGNAL_DCMIDA_11-	D26	-A429_RX00 (RESERVED)	USER DEFINED SIGNAL_+5VExcA_1-
26	USER DEFINED SIGNAL_DCMIDA_12+	USER DEFINED SIGNAL_DCMIDA_12-	D27	422A_TX+0 (RESERVED)	USER DEFINED SIGNAL_+5VExcA_2+
27	USER DEFINED SIGNAL_DCMIDA_13+	USER DEFINED SIGNAL_DCMIDA_13-	D28	422A_TX-0\232A_Tx0 (RESERVED)	USER DEFINED SIGNAL_+5VExcA_2-
28	USER DEFINED SIGNAL_DCMIDA_14+	USER DEFINED SIGNAL_DCMIDA_14-	D29	422_RX+0 (RESERVED)	USER DEFINED SIGNAL_+10VExcA_1+
29	USER DEFINED SIGNAL_DCHIA_1+	USER DEFINED SIGNAL_DCHIA_1-	D30	422_RX-0\232_Rx0 (RESERVED)	USER DEFINED SIGNAL_+10VExcA_1-
30	USER DEFINED SIGNAL_DCHIA_2+	USER DEFINED SIGNAL_DCHIA_2-	D31	Serial Format	USER DEFINED SIGNAL_+10VExcA_2+
31	USER DEFINED SIGNAL_DCHIA_3+	USER DEFINED SIGNAL_DCHIA_3-	DGND	Analog A Reprogram	USER DEFINED SIGNAL_+10VExcA_2-
32	USER DEFINED SIGNAL_DCHIA_4+	USER DEFINED SIGNAL_DCHIA_4-	+5V	USER DEFINED SIGNAL_SpareA Reset	AGND

Table B - 3 Spare Board A, JA2 Pin Outs

JB1

PIN	ROW A	ROW B	ROW C
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	DGND	BG2OUT*	DGND
10	SYSCLK	BG3IN*	SYSFAIL*
11	DGND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	DGND	BR3*	A23
16	DTACK*	AM0	A22
17	DGND	AM1	A21
18	AS*	AM2	A20
19	DGND	AM3	A19
20	IACK*	DGND	A18
21	IACKIN*	SERCLK	A17
22	IACKOUT*	SERDAT*	A16
23	AM4	DGND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	NC (+5V)	+12V
32	+5V	+5V	+5V

Table B - 4 Spare Board B, JB1 Pin Outs

JB0

PIN	ROW A	ROW B	ROW C	ROW D	ROW E
1	USER DEFINED SIGNAL_ACCoupB_1 a+	USER DEFINED SIGNAL_ACCoupB_1 a-	AGND	USER DEFINED SIGNAL_DisB_22	USER DEFINED SIGNAL_DisB_41
2	USER DEFINED SIGNAL_ACCoupB_1 b+	USER DEFINED SIGNAL_ACCoupB_1 b-	USER DEFINED SIGNAL_DisB_7	USER DEFINED SIGNAL_DisB_23	USER DEFINED SIGNAL_DisB_42
3	USER DEFINED SIGNAL_ACCoupB_1 c+	USER DEFINED SIGNAL_ACCoupB_1 c-	USER DEFINED SIGNAL_DisB_8	USER DEFINED SIGNAL_DisB_24	USER DEFINED SIGNAL_DisB_43
4	USER DEFINED SIGNAL_ACCoupB_1 d+	USER DEFINED SIGNAL_ACCoupB_1 d-	USER DEFINED SIGNAL_DisB_9	USER DEFINED SIGNAL_DisB_25	USER DEFINED SIGNAL_DisB_44
5	USER DEFINED SIGNAL_ACCoupB_2 a+	USER DEFINED SIGNAL_ACCoupB_2 a-	USER DEFINED SIGNAL_DisB_10	USER DEFINED SIGNAL_DisB_26	USER DEFINED SIGNAL_DisB_45
6	USER DEFINED SIGNAL_ACCoupB_2 b+	USER DEFINED SIGNAL_ACCoupB_2 b-	USER DEFINED SIGNAL_DisB_11	USER DEFINED SIGNAL_DisB_27	USER DEFINED SIGNAL_DisB_46
7	USER DEFINED SIGNAL_ACCoupB_2 c+	USER DEFINED SIGNAL_ACCoupB_2 c-	USER DEFINED SIGNAL_DisB_12	USER DEFINED SIGNAL_DisB_28	USER DEFINED SIGNAL_DisB_47
8	USER DEFINED SIGNAL_ACCoupB_2 d+	USER DEFINED SIGNAL_ACCoupB_2 d-	USER DEFINED SIGNAL_DisB_13	USER DEFINED SIGNAL_DisB_29	USER DEFINED SIGNAL_DisB_48
9	USER DEFINED SIGNAL_ACCoupB_3 a+	USER DEFINED SIGNAL_ACCoupB_3 a-	USER DEFINED SIGNAL_DisB_14	USER DEFINED SIGNAL_DisB_30	USER DEFINED SIGNAL_FreqB_1+
10	USER DEFINED SIGNAL_ACCoupB_3 b+	USER DEFINED SIGNAL_ACCoupB_3 b-	USER DEFINED SIGNAL_DisB_15	USER DEFINED SIGNAL_DisB_31	USER DEFINED SIGNAL_FreqB_1-
11	USER DEFINED SIGNAL_ACCoupB_3 c+	USER DEFINED SIGNAL_ACCoupB_3 c-	USER DEFINED SIGNAL_DisB_16	USER DEFINED SIGNAL_DisB_32	USER DEFINED SIGNAL_FreqB_2+
12	USER DEFINED SIGNAL_ACCoupB_3 d+	USER DEFINED SIGNAL_ACCoupB_3 d-	USER DEFINED SIGNAL_DisB_17	USER DEFINED SIGNAL_DisB_33	USER DEFINED SIGNAL_FreqB_2-
13	USER DEFINED SIGNAL_ACCoupB_4 a+	USER DEFINED SIGNAL_ACCoupB_4 a-	USER DEFINED SIGNAL_DisB_18	USER DEFINED SIGNAL_DisB_34	USER DEFINED SIGNAL_FreqB_3+
14	USER DEFINED SIGNAL_ACCoupB_4 b+	USER DEFINED SIGNAL_ACCoupB_4 b-	USER DEFINED SIGNAL_DisB_19	USER DEFINED SIGNAL_DisB_35	USER DEFINED SIGNAL_FreqB_3-
15	USER DEFINED SIGNAL_ACCoupB_4 c+	USER DEFINED SIGNAL_ACCoupB_4 c-	USER DEFINED SIGNAL_DisB_20	USER DEFINED SIGNAL_DisB_36	USER DEFINED SIGNAL_FreqB_4+
16	USER DEFINED SIGNAL_ACCoupB_4 d+	USER DEFINED SIGNAL_ACCoupB_4 d-	USER DEFINED SIGNAL_DisB_21	USER DEFINED SIGNAL_DisB_37	USER DEFINED SIGNAL_FreqB_4-

17	USER DEFINED SIGNAL_DisB_1	USER DEFINED SIGNAL_DisB_4	+28V Reg	USER DEFINED SIGNAL_DisB_38	USER DEFINED SIGNAL_FreqB_5+
18	USER DEFINED SIGNAL_DisB_2	USER DEFINED SIGNAL_DisB_5	+28V Reg Rtn	USER DEFINED SIGNAL_DisB_39	USER DEFINED SIGNAL_FreqB_5-
19	USER DEFINED SIGNAL_DisB_3	USER DEFINED SIGNAL_DisB_6	AGND	USER DEFINED SIGNAL_DisB_40	USER DEFINED SIGNAL_A/B Discrete

Table B - 5 Spare Board B, JB0 Pin Outs

JB2

PIN	ROW Z	ROW A	ROW B	ROW C	ROW D
1	DCLOWB_1+	DCLOWB_1-	+5V	Low Hold Up	AGND
2	DCLOWB_2+	DCLOWB_2-	DGND	IOatPrB_1+	IOatPrB_1-
3	DCLOWB_3+	DCLOWB_3-	RESERVED	IOtPrbB_2+	IOtPrbB_2-
4	DCLOWB_4+	DCLOWB_4-	A24	IDCExcB_1+	IDCExcB_1-
5	DCLOWB_5+	DCLOWB_5-	A25	IDCExcB_2+	IDCExcB_2-
6	DCLOWB_6+	DCLOWB_6-	A26	IDCExcB_3	IDCExcB_3-
7	DCLOWB_7+	DCLOWB_7-	A27	IDCExcB_4+	IDCExcB_4-
8	DCLOWB_8+	DCLOWB_8-	A28	Vbat	CJTB+
9	DCLOWB_9+	DCLOWB_9-	A29	DCCoupB_1a+	+15VB
10	DCLOWB_10+	DCLOWB_10-	A30	DCCoupB_1a-	-15VB
11	DCLOWB_11+	DCLOWB_11-	A31	DCCoupB_1b+	DCCoupB_1b-
12	DCLOWB_12+	DCLOWB_12-	DGND	DCCoupB_1c+	DCCoupB_1c-
13	DCLOWB_13+	DCLOWB_13-	+5V	DCCoupB_1d+	DCCoupB_1d-
14	DCLOWB_14+	DCLOWB_14-	D16	GENSHUBF_1+	GENSHUBF_1-
15	DCMIDB_1+	DCMIDB_1-	D17	GENSHUBF_2+	GENSHUBF_2-
16	DCMIDB_2+	DCMIDB_2-	D18	HDLB+	HDLB-
17	DCMIDB_3+	DCMIDB_3-	D19	DISOUTB_1+	DISOUTB_1-
18	DCMIDB_4+	DCMIDB_4-	D20	DISOUTB_2+	DISOUTB_2-
19	DCMIDB_5+	DCMIDB_5-	D21	DISOUTB_3+	DISOUTB_3-
20	DCMIDB_6+	DCMIDB_6-	D22	DISOUTB_4+	DISOUTB_4-
21	DCMIDB_7+	DCMIDB_7-	D23	ACExcB_1+	ACExcB_1-
22	DCMIDB_8+	DCMIDB_8-	DGND	+A429_TX00	ACExcB_2+
23	DCMIDB_9+	DCMIDB_9-	D24	-A429_TX00	ACExcB_2-
24	DCMIDB_10+	DCMIDB_10-	D25	+A429_RX01	+5VExcB_1+
25	DCMIDB_11+	DCMIDB_11-	D26	-A429_RX01	+5VExcB_1-
26	DCMIDB_12+	DCMIDB_12-	D27	422B_TX+0	+5VExcB_2+
27	DCMIDB_13+	DCMIDB_13-	D28	422B_TX-0\232B_Tx0	+5VExcB_2-
28	DCMIDB_14+	DCMIDB_14-	D29	422_RX+0	+10VExcB_1+
29	DCHIBF_1+	DCHIBF_1-	D30	422_RX-0\232_Rx0	+10VExcB_1-
30	DCHIBF_2+	DCHIBF_2-	D31	Serial Format	+10VExcB_2+
31	DCHIB_3+	DCHIB_3-	DGND	Analog B Reprogram	+10VExcB_2-
32	DCHIB_4+	DCHIB_4-	+5V	Spare B Reset	AGND

Table B - 6 Spare Board B, JB2 Pin Outs

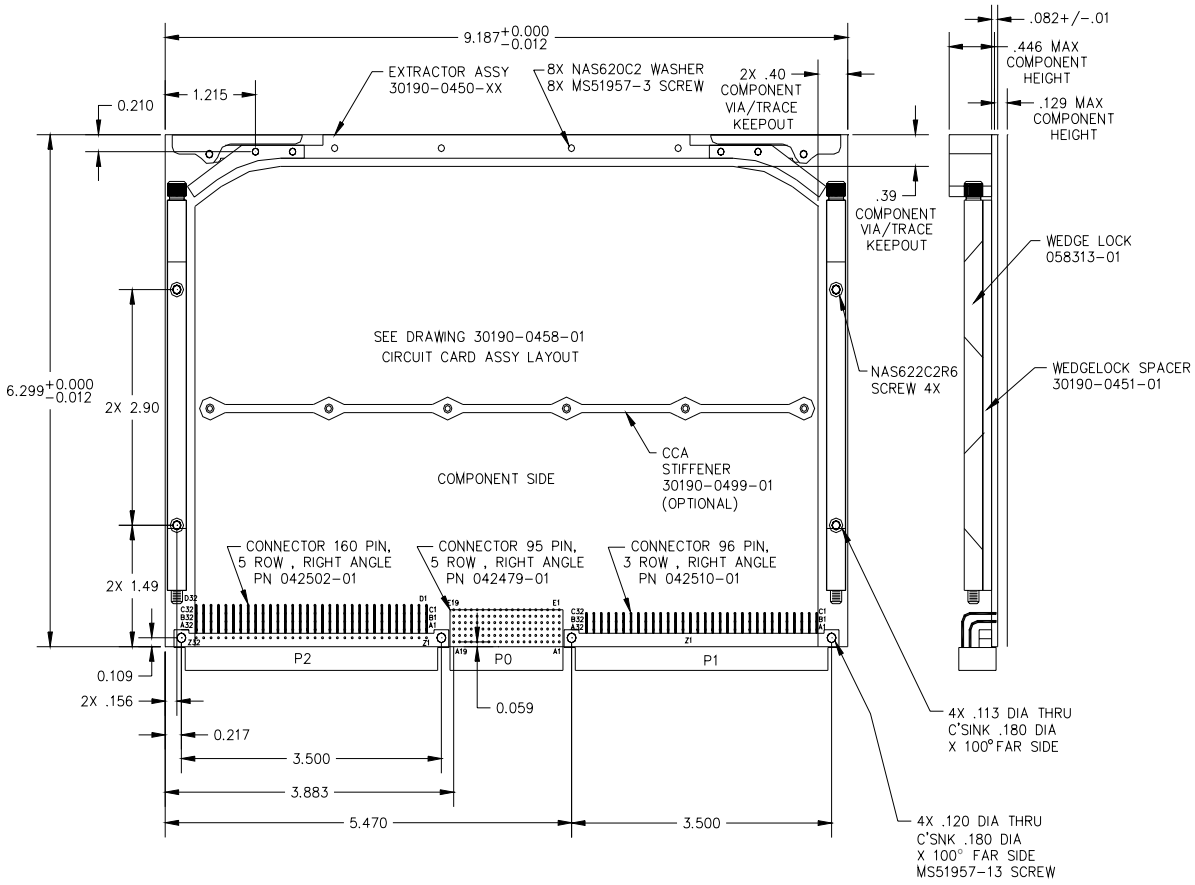


Figure B - 2 HUMS MPU Spare Slot Geometry

Appendix C - Technology Integration Questionnaire

1 Objectives

This document is intended to collect source information necessary to perform the integration of a technology module into Goodrich's Integrated Mechanical Diagnostics Health and Usage Monitoring System (IMD HUMS). The intent is to use this information to create a Technology Integration Requirements Specification. Whereas the Open Systems Requirements Specification (T1009-0100-0101) provides a detailed definition of the available interfaces and capabilities of the IMD HUMS. The Technology Integration Requirements Specification shall: detail resource requirements, define the interfaces in sufficient detail to allow the generation of Configuration Tables, provide Built in Test (BIT) definition and isolation methods, define the integration process, specify validation and qualification methods specific for each technology insertion project. Because the IMD HUMS is an Open System, the information contained in Technology Integration Specification is not proprietary to any party.

2 Main Processor Resource Requirements

2.1 Hardware Resources

The Main Processor Unit can accommodate two hardware modules that comply with the VME 6U standard.

2.1.1 Power Supply Resources

Complete the following table by indicated the required current for each available voltage. Indicate any special requirements or considerations in the comment column.

Voltage (Available)	Current	Comments
5 Volts		
+15 Volts		
-15 Volts		
28 Volts		

2.1.2 Power Status

Does the technology module have any special power up requirements?
Does the technology module require any special power hold-up or power down processing?

2.1.3 Power Dissipation

What is the total internal power dissipation of the technology module?

2.1.4 VME Card Slot External Signal I/O

There are four rows of the P2 VME connector that have been connected to the external MPU ARINC-600 connector. Describe the external I/O requirements of the technology module in general terms (type of signals, quantity, purpose, required excitation, equipment specification, wiring requirements, pairing of signals etc.)?

Complete the following tables for each interface signal.

Signal Name	Quantity	Range	Accuracy	Resolution	Source Impedance	Signal Reference	Interface Req.

Signal Name	Signal Type (i.e. voltage, current, etc)	Input Impedance	Bandwidth	Isolation Req.	Fault Voltages, currents

2.1.5 VME Bus Interface

The Primary Processor Unit provides VME bus control. Technology modules on the VME bus have read write access to data.

Describe VME bus message throughput requirements (Information, rates, integrity etc).

2.1.6 Environmental Requirements.

Describe environmental requirements that the technology module has been or will be qualified to. Describe assumptions associated with these qualification levels. Describe (cooling assumptions such as conduction/convection. Describe vibration assumptions such as chassis transmissibility. Are there any known limitations such as vibration or temperature environments?

2.1.7 Electromagnetic Computability Requirements

Describe the electromagnetic computability requirements associated with the technology module. Are there any special considerations or limitations associated with the module or interface signals?
A filter pin connector accomplishes the EMIC to external environments. Are there any considerations/ requirements associated with each external signal type?

Internal EMIC is accomplished by isolation, shielding and ground planes. Describe any special considerations associated with each signal type?

2.2 Primary Processor Unit

2.2.1 PPU Embedded Software

Is there a need to integrate Ada software into the Primary Processing Environment?
If so complete the following table for each module:

Module	SLOC	Memory Reqs. Prog./Data	Execution Rate	Execution Req.	Execution Priority	Execution Time	Comments

2.2.2 PPU Data Repository Resource Requirements

Is there a need to access core aircraft parameters? If so specify a list of repository nomenclature.
Is there a need to create additional Repository Data? If so specify.

- **Describe the Repository Items to be defined (for each) ::**

  Item Name : {Follows Syntax for Ada Identifiers}
  Item Type : {Float_32 | Signed_Integer_32 | Unsinged_Integer_32 | Boolean}
  Persistent : {Yes | No}
  Number of Instances : {1..132 if not persistent, 1 if persistent}

Note: Persistence means that the value is saved across power cycles.

2.2.3 Basic Actions

Does the technology module require the derivation of data, calculation of exceedances, of the recognition of regimes by the PPU?

2.2.4 Procedural Actions

Does the technology module require the sequencing of a set of actions by the PPU? If so then consider the following questions

- **Explain the Action Procedures 'Entrance Criteria'**.

Do you want the entrance criteria to be a simple Boolean Value that YOU set true via your PPU resident code?
Do you want the entrance criteria to be one of the predefined Goodrich Regimes? If yes which one?
Do you want the entrance criteria to be a set of Boolean expressions involving Goodrich Core Parameters? If yes what are the expressions?

- **Explain the VPU Data Acquisition and Data Processing Duration?**

What is the time duration of the VPU acquisition phase?
What is the duration of the VPU processing phase?

- **Do you want to abort the Action Procedure during the VPU acquisition phase if a specific set of criteria is not met? If so, specify the criteria?**

Do you want the criteria to be a simple Boolean Value that YOU set via your PPU resident code?
Do you want the criteria to be one of the predefined Goodrich Regimes? If yes which one?
Do you want the criteria to be a set of Boolean expressions involving Goodrich Core Parameters? If yes what are the expressions?

2.2.5 Event Detection

Does the technology module require the detection of events? If so what actions need to be taken upon the detection of this event by the PPU?

2.2.6 Interrupt

Does the technology module require the processing of interrupts? If so describe.

2.3 Vibration Processor Unit

2.3.1 VPU Embedded Software

Is there a need to integrate software into the Vibration Processing Environment?
If so complete the following table for each module:

Module	SLOC	Memory Reqs. Prog./Data	Execution Rate	Execution Req.	Execution Priority	Execution Time	Comments

2.3.2 Accelerometer Acquisition Requirements

Does the technology module require the processing of accelerometers? If so complete the following table.

Accel Name	Qty	Bandwidth	Characteristics	Acquisition time.	Sync Req.	Trigger Req	Comments

2.4 Main Processor Unit Signal Conditioning Resource Requirements

The Main Processor Unit has the capability to interface with the type of signals indicated by the following table. Are any of these types of interfaces required? If so complete the information requested.

Signal Name	Quantity	Range	Accuracy	Resolution	Source Impedance	Signal Reference	Interface Req.
Freq. Input							
Accelerometers							
Tach Signals							
Optical Tracker							
Index sensors							
Discretes							
Cockpit audio							
Synchro Signals							

2.5 Main Processor Serial Interface Requirements

Does the technology module require information from a serial bus? If so what type of serial device (Mil-std-1553B, ARINC 429, RS-422, ARINC 717)?

Describe the parameters that need to be transferred. Describe the throughput and update requirements.

3 Input Output Requirements

3.1 Remote Data Concentrator Requirements

The Remote Data Concentrator provides the capability to convert analog and digital signals onto an ARINC 429 serial bus. Does the technology module require this capability?

If so complete the following tables for each interface signal.

Signal Name	Quantity	Range	Accuracy	Resolution	Source Impedance	Signal Reference	Interface Req

Signal Name	Signal Type (i.e. voltage, current, etc)	Input Impedance	Bandwidth	Isolation Req.	Fault Voltages, currents

3.2 Data Transfer Requirements

The Data Transfer Unit (DTU) provides the capability of transferring information between the MPU and the Ground Support Station (uploading configuration tables and downloading information collected and processed during flight. Does the technology module require this capability?

3.2.1 Upload Requirements

Does the technology module require the upload of information? If so describe the type, frequency and quantity of this information.

3.2.2 Download Requirements

Does the technology module require the download of information? If so describe the type and quantity of this information. What is the rate of this transfer? What are the memory requirements?

3.3 Aircrew User Interface Requirements

Does the technology module require the display of information to the aircrew? If so describe the displays using the layout aids below.

The following is a diagram that represents a typical display on an aircraft using the Goodrich CDU. Although the actual display may change for each aircraft model, the capabilities remain similar. Contact Goodrich for the capabilities available on a particular aircraft.

Full Capability

Lines 00 through 15
Columns 00 through 23

Usable

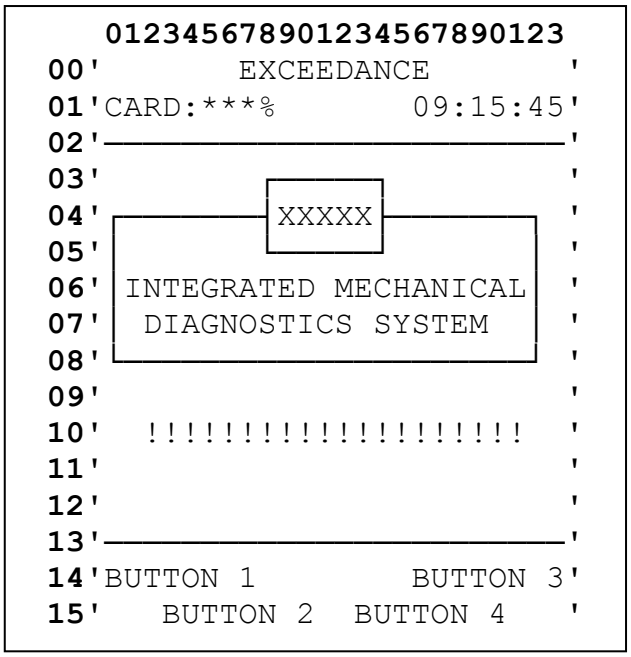
On Line 00 and Line 15
Only Columns 04 through 19

On Lines 01 through 14
Columns 00 through 23

Typical Usage

Lines 03 through 12
Columns 00 through 23

Line 14 Button Labels Only
Line 15 Button Labels Only



Line Drawing Characters Available

┘ = Lower Right Corner
┐ = Upper Right Corner
┌ = Upper Left Corner
└ = Lower Left Corner
│ = Vertical Bar
─ = Horizontal Bar
┼ = Cross Bar
┴ = Upside Down T
┬ = Normal T
├ = Left T
┤ = Right T

Fig. 1 Display Screen and Layout

The screen shows a typical display on the aircraft. The numbers along the side and top are not displayed; they are present to serve as a visual aid. The line drawing characters shown to the right are used to partition the display into specific areas. They can also be used to build crude graphics and animations. Caution, the displays are usually dumb in nature and as such can require a great deal of CPU processing to display animations by switching between several pages of graphics.

- **Screen Definition Worksheet**

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
00																								
01																								
02																								
03																								
04																								
05																								
06																								
07																								
08																								
09																								
10																								
11																								
12																								
13																								
14																								
15																								

Note: The blackened boxes are not displayable on the CDU.
The grayed boxes are reserved for use by Goodrich

Do you want to display any repository items ?

Repository Item Name :
Repository Item Number :
Starting Line # :
Starting Column # :

Does the technology module require aircrew interaction? If so describe.

What action do you wish to perform when Button 1 is pressed?

What action do you wish to perform when Button 2 is pressed?

What action do you wish to perform when Button 3 is pressed?

What action do you wish to perform when Button 4 is pressed?

4 Ground Support Station Resource Requirements

4.1 Platform Requirements

Describe the hardware software requirements of the technology module.

What NT registry settings are required?
What are the memory requirements?
How many software components (dll, exe, ….)?
Does the technology module require any support files?

4.2 ADF

Will the technology module require access to information in the ADF? If so describe.

4.3 NALCOMIS Database Resource Requirements

Will the technology module require access to information contained in the NALCOMIS Database? If so describe.

4.4 User Interface

4.4.1 Icon Launch

The GSS has the capabilities to Icon launch technology modules. Is this capability desired? If so provide the interface information required to launch the module. What is the present software interface?

4.4.2 Graphical User Interface

Describe and user interface requirements.
What is the GUI technology used (Visual C++, Visual Basic, etc)?

5 Built in Test Methods

If the technology module consists of hardware components, describe BIT philosophies that will allow isolation of failures to either the baseline system or the technology module.
Describe failure modes and effects of the technology.
Describe how each of these failure modes would be detected.

6 Integration and Test Requirements Plan

Describe recommended integration methods for the technology module.
Describe test methods that will be used to verify the functional operation of the technology module prior to integration.
Describe test requirements to verify integration of the technology module.
Describe special test equipment necessary to verify operation of the technology module.
Describe required test cases.
Describe product Acceptance Test requirements.
Describe field test requirements.

7 Validation Requirements

Describe any requirements associated with validating the technology module. Consider all phases of integration, bench, A/C prototype, and A/C fleet.

8 Qualification Requirements

Describe test methods that will be used to qualify the technology module.

-Hardware
-Software